AN UHF FMCW

WIND PROFILER DEVELOPMENT AND RESULTS

A Thesis Presented By

Carolina Asturiano Gómez

Projecte Final de Carrera

Enginyeria de Telecomunicació

Submitted to Universitat Politècnica de Catalunya

(Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona)

in fulfillment of the requirements for:

Projecte Final de Carrera

Enginyeria de Telecomunicació

Approved as to style and content by:

Stephen J. Frasier, Advisor

Amherst, Massachusetts
May 2010

In any situation,

the best thing you can do is the right thing;

the next best thing you can do is the wrong thing;

the worst thing you can do is nothing.

Theodore Roosevelt

ABSTRACT

Radars based on the Frequency-Modulated Continuous-Wave (FMCW) technique have proven to be an excellent tool for the Atmospheric Boundary Layer (ABL) studies. This layer of the atmosphere is of vital importance for the understanding of weather and climate. FMCW radars provide high sensitivity and high range resolution needed for observation of small scale turbulence.

This thesis describes the FMCW Wind Profiler operating at UHF band as of Spring 2010. At a later stage it will utilize the spaced antenna (SA) technique for achieving three-dimensional wind fields. This work describes its principles of operation, the current status of the radar hardware, and comments the new programs developed for its proper operation. It discusses the sensitivity of the Wind Profiler derived from laboratory tests and the results of the field deployments. The current design uses four antennas, one for transmission and three for reception. The major drawback of the Wind Profiler is the isolation required between the transmitter and the receiver. A new scheme is proposed to overcome this problem by using Interrupted FMCW. Recommendations for the future hardware development of the radar and conclusions are provided.

TABLE OF CONTENTS

ABSTRACT iv

LIST OF FIGURES AND TABLES viii

LIST OF SYMBOLS xi

ACKNOWLEDGEMENTS xii

CHAPTER 1 INTRODUCTION 1

1.1 Motivation 1

1.2 Summary of Chapters 3

CHAPTER 2 WIND PROFILER RADAR BACKGROUND 5

2.1 Atmospheric boundary layer 5

2.2 Clear air back scattering Theory 6

2.3 FMCW Principles 7

2.4 Interrupted FMCW 12

CHAPTER 3 SYSTEM DESCRIPTION 15

3.1 Current Hardware Configuration 15

3.2 Windprofiler Subsystems 18

3.2.1 Control and Transmit Subsystem 18

3.2.2 Calibration Loop 19

3.2.3 Receiver Subsystem 19

CHAPTER 4 DATA ACQUISITION 21

4.1 Acquisition Board 21

4.2 Data acquisition program 24

CHAPTER 5 SIGNAL PROCESSING 27
5.1 Signal processing programs 27
5.2 GUI 29
CHAPTER 6 RADAR TESTS 31
6.1 Synchronization 31
6.2 Minimum detectable Signal 33
CHAPTER 7 FIELD DEPLOYMENTS 41
7.1 Antenna Isolation 41
7.2 Deployment results 42
CHAPTER 8 FUTURE WORK 51
8.1 FMICW configuration 51
8.2 Spaced Antenna 54
CHAPTER 9 CONCLUSIONS 57
BIBLIOGRAPHY 59
APPENDIX A - ACQUISITION SOURCE CODE 61
A.1 Savedata_Windprof 61
A.1.1 main.c 61
A.1.2 savedata.c 65
A.1.3 fmcw_conf.c 69
APPENDIX B - SIGNAL PROCESSING PROGRAMS 73
B.1 Windprofiler2010 73
B.1.1 Windprof_process.pro 74
B.1.2 Windprof_process2.pro 77

B.1.3 Windprof_display.pro 77
B.1.4 Windprof_process_realtime.pro 79
B.1.5 Windprof_display_realtime.pro 82
B.1.6 GUI 86
B.1.7 wgui_create.pro 86
B.1.8 wgui_event.pro 88
APPENDIX C - RADAR BOXES 93
C.1 Boxes 93
C.2 Connections 94

LIST OF FIGURES AND TABLES

Figure 2.1 The diurnal evolution of the atmospheric boundary layer over land (from Wyngaard, 1992) 5

Figure 2.2 Waveform of sent signal 8

Figure 2.3 FMCW Radar Sweep (left) and Frequency diagram (right) 9

Figure 2.4 FMICW principles 12

Figure 2.5 FMICW Sensitivity function 13

Figure 3.1 Current FMCW Wind Profiler block diagram 17

Figure 3.2 FPGA Control signals generation 19

Figure 3.3 Audio module filter response 20

Figure 4.1 24DSI6LN Data Acquisition Card 21

Figure 4.2 DAQ Connector 23

Figure 5.1 Amplitude of a chirp and Power Spectrum 28

Figure 5.2 FMCW radar GUI 30

Figure 6.1 Power spectrum (left) and Phase of the 300m bin and 400m bin (right)31

Figure 6.2 Power spectrum (left) and Phase of the 300m bin (right) with phase correction 32

Figure 6.3 Minimum detectable with default configuration 34

Figure 6.4 Lab Setup 35

Figure 6.5 Noise floor with and without 17dB amplifier 35

Figure 6.6 Noise floor with averaging of 1000 profiles 36

Figure 6.7 Noise floor adding 2 profiles 36

Figure 6.8 Minimum detectable C_n^2 when received power is -160dBm 37

Figure 6.9 Noise floor with different configurations (Averaging of 1000 profiles) 39

Figure 6.10 Noise floor $P_{leak} = -40.5dBm$ 40

Figure 7.1 Antennas in Tillson farm 41

Figure 7.2 Received leakage 42

Figure 7.3 Minimum detectable C_n^2 when received power is -140dBm 43

Figure 7.4 Returned power and Doppler velocity during deployment on April 14th (Default configuration) 45

Figure 7.5 Returned power and Doppler velocity during deployment on April 14th (B=12.5MHz) 45

Figure 7.6 Returned power and Doppler velocity during deployment on May 5th 46

Figure 7.7 Data Power Spectrum (April 14th) 46

Figure 7.8 Returned power and Doppler velocity on April 22nd 47

Figure 7.9 Audio module output 47

Figure 7.10 Antennas with additional shrouds 48

Figure 7.11 Returned power and Doppler velocity during deployment on May 12th49

Figure 8.1 FMICW Block Diagram 51

Figure 8.2 3-antenna SA-FMICW 54

Figure C.1 Wind Profiler Boxes 93

Figure C.2 Radar connections 94

Table 3.1 System specifications 16

Table 4.1 Input Data Buffer Organization 22

Table 4.3 Files in `savedata_windprof/` 24

Table 5.1 FMCW Radar collection&display data procedure 29

Table 6.1 FMCW parameters 38

Table 8.1 High Power Switch SPDT (Ref. ES0309-50) 52

Table 8.2 Low Power Switch SPST (Ref. ZFSWHA-1-20+) 53

Table 8.3 FMICW parameters 54

Table B.1 Windprofiler2010 files 73

LIST OF SYMBOLS

Structure Constant . . . . . C_n^2

Sweep Time . . . . . T_P

Bandwidth . . . . . B

Range . . . . . R

Beat frequency . . . . . f_b

Sampling frequency . . . . . f_{Sample}

Switch frequency . . . . . f_S

Range resolution . . . . . $\triangle R$

Compression gain . . . . . G_C

Transmit power . . . . . P_t

Receive power . . . . . P_r

Transmit antenna gain . . . . . G_t

Receive antenna gain . . . . . G_r

Radar wavelength . . . . . λ

Half power beamwidth . . . . . θ

Volume reflectivity . . . . . η

Antenna correction term . . . . . C_a

ACKNOWLEDGEMENTS

Firstly, I would like to thank the supervisor of this thesis, Dr. Stephen Frasier for his assistance and supervision during the whole process of this work, and also for giving me the opportunity to be at MIRSL. It has been an excellent experience that really helped me to expand my knowledge a lot, and to grow as an engineer. I would like to thank to all the people working here at MIRSL who were always willing to answer my questions and whenever I needed, they helped me, even in the field deployments. I would especially give thanks to David Garrido, who initially walked me through the radar project. Without his help this work would have been harder. Also thanks to him for being a good friend, making me feel comfortable since my first day in Amherst.

I am also grateful to the friends with whom I have been sharing all these time as an exchange student here in Amherst. Thanks to them, these months are unforgettable. Finally, my warmest thanks to my friends from back home, my boyfriend and my family for supporting me from a great distance.

CHAPTER 1

INTRODUCTION

1.1 Motivation

The Atmospheric Boundary Layer (ABL) is the lowest few hundred meters of the atmosphere, where the Earth's surface most directly influences atmospheric processes. The structure and dynamics of the ABL are of vital importance for the understanding of weather and climate, the dispersion of pollutants, and the exchange of heat, water vapor, and momentum with the underlying surface. The knowledge of this it allows more refined and reliable numerical models for daily weather forecasts, dispersion events, and future global climate on Earth. The effect of the troposphere on wave propagation has also been studied extensively for improving radio communications. Radio signals are altered by slight changes in the refractive index in the lower atmosphere caused by air turbulence, and small changes in temperature, humidity and barometric pressure. This makes it very important to have a full and detailed description of the refractive index structure.

Frequency modulated continuous wave (FMCW) radars are able to provide information on the ranges and Doppler shifts of multiple targets. This information is essentially equivalent to that provided by coherent pulsed radars. However, as the signal to noise performance of a radar depends on average, rather than peak transmitted power, the high duty cycle of the FMCW waveform can provide good performance from a relatively compact and inexpensive radar system. Additional advantages come from the fact that the range resolution of an FMCW radar is determined principally by the bandwidth of the linear frequency sweep. Consequently, high range resolutions can be obtained without the attendant problems of generating very short pulses. Further, the FMCW waveform provides good performance at close ranges and low interference to other systems when compared with conventional pulsed

radars. A major disadvantage of FMCW systems is the need to use two separate, well shielded antennas for transmission and reception.

There exists an S-band FMCW radar for atmospheric boundary layer studies which was developed in The Microwave Remote Sensing Laboratory (at University of Massachusetts, Amherst) [1]. This radar is sensitive to Rayleigh scattering from insects which can be much greater than the weak Bragg scattering from refractive index irregularities. Therefore it was proposed to change the frequency from S-band (3GHz) to UHF (915MHz). Vertically pointed radars can measure vertical wind velocities. In order to retrieve horizontal winds, the proposed profiler uses a spaced antenna (SA).

A low-power, cost effective and mobile UHF FMCW-SA Wind profiler Radar is being developed at MIRSL. It will allow to observe and retrieve information about the structure and processes in the ABL with high resolution. The Wind Profiler design and development was initiated in the summer of 2006 by Prof. Frasier and the former students Albert Genis during 2008, Iva Kostadinova on 2008-2009 and David Garrido on 2009. Many changes have been introduced to the radar, after upgrading and overcoming the different problems that were affecting it.

The primary objective of this thesis is to continue with the work achieved by David Garrido [2,3], first of all focusing on data acquisition and synchronization issues, and secondly developing the required signal processing programs. The current hardware configuration of the UHF Wind Profiler, as well as the obtained outcomes, will be described in detail in the laboratory and the field deployment. Finally, recommendations for future work are discussed.

1.2 Summary of Chapters

This thesis documents the FMCW Wind Profiler radar as of Spring 2010. Chapter 2 provides a brief description of the atmospheric boundary layer, discusses the principles of operation of FMCW radars as well as Interrupted FMCW radars. Chapter 3 provides an overview of the current status of the Wind Profiler hardware.

The main part of this thesis begins with the development of the new data acquisition programs, and subsequently, the development of the signal processing programs required for the proper operation of the radar. These are described in Chapters 4 and 5 respectively. A series of tests are performed in the laboratory for the evaluation of the Wind Profiler. The adopted solution for the synchronization issues, and the minimum detectable signal for the Wind Profiler are discussed in Chapter 6. In the next chapter, the results of the field deployment are presented. Chapter 8 comments some future work that can be done in the system, a new scheme is proposed to enable the radar to operate with a single antenna for transmission and reception. Finally, Chapter 9 concludes this thesis.

CHAPTER 2

WIND PROFILER RADAR BACKGROUND

2.1 Atmospheric boundary layer

The atmospheric boundary layer is the lower part of the atmosphere that is directly influenced by the surface of the Earth. The diurnal heat, moisture or momentum transfer to or from the surface create strong mixing and turbulent processes with a time scales of about an hour or less. The structure of ABL turbulence is strongly affected by the diurnal cycle of surface heating and cooling, and by presence of clouds. Basically there are two types of boundary layer based on buoyancy effects: (i) Convective boundary layer, where heat from the surface of the Earth creates positive buoyancy flux and instabilities that lead to turbulence and (ii) Nocturnal boundary layer, where negative buoyancy flux decreases the turbulence and stable stratified conditions prevail.

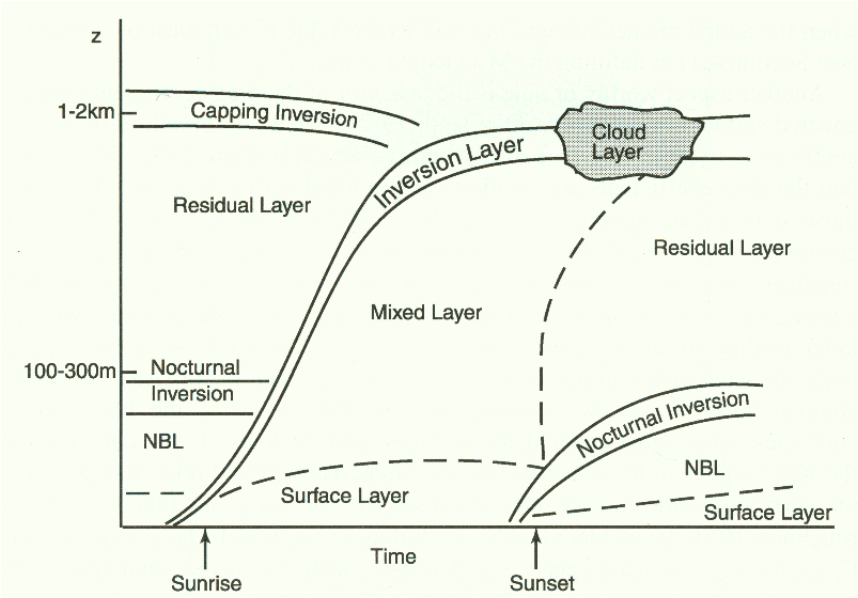


Figure 2.1: The diurnal evolution of the atmospheric boundary layer over land (from Wyngaard, 1992)

The nocturnal boundary layer may also be convective when cold air advects over a

warm surface. Typical boundary layer structure is depicted in figure Figure 2.1. The depth of the ABL can range from only a few meters in strongly stable stratification to several kilometers in highly convective conditions.

2.2 Clear air back scattering Theory

In 1966 Hardy and Atlas were the first to attribute the detected backscattered signals from clear air due to Bragg scattering from random fluctuations in the refractive index [4]. Radar backscattering from refractive index variation is able to provide helpful information about the atmospheric structure.

The radar backscattering from the clear air atmosphere, is caused by irregular small-scale fluctuations in the radio refractive index produced by turbulent mixing and is dependent on the atmospheric water vapor, temperature, and pressure.

The parameter C_n^2 is called the structure constant and is a major characteristic describing the intensity of atmospheric turbulence. In 1969 Ottersen [5] was the first to derive the relationship between the radar volume reflectivity and refractive index structure as

$$\eta(\lambda) \approx 0.38 C_n^2 \lambda^{-\frac{1}{3}} \tag{2.1}$$

For a given radar , the radar reflectivity of a region of refractive index fluctuations is directly proportional to C_n^2 when the length scale of one-half of the radar wavelength falls within the inertial subrange. The more violent the turbulent mixing the larger the displacements, and the stronger the inhomogeneities will be. Consequently, strong turbulence and sharp mean gradients contribute to high C_n^2 values. Typical values of the structure constant for the convective boundary layer lie in the range between 10^{-15} to $10^{-13} m^{\frac{-2}{3}}$. Radar backscatter can also be due to Rayleigh scattering from birds, insects, and hydrometeors where the size of the scatterers is much smaller than the radar wavelength.

2.3 FMCW Principles

FMCW is a common technique used in radar technology, that is not limited by the constraints of pulsed radars. FMCW consists of the transmission of a sinusoid whose frequency changes over time. FMCW Radar transmits a frequency sweep, commonly called a chirp. The signal is reflected from distant targets and detected by the receiver where the returned signal is mixed with a copy of the transmitted signal to determine the range of the target. The transmitted waveform has a time varying frequency $f(t)$ given by:

$$f(t) = F_1 + \alpha \cdot t \quad \forall t < T_P \tag{2.2}$$

where F_1 is the initial frequency, α the rate of change of frequency and T_P the Sweep time. This is a linearly increasing frequency sweep as shown in Figure 2.2. Ideally the instantaneous frequency should augment indefinitely with time, but in order to have a realistic system the frequency will increase until a maximum value, and will start from the initial frequency once that point is reached. The frequency increases from F_1 at the start of the sweep to F_2 at the end of the sweep after a time T_P. The bandwidth B is the difference between F_2 and F_1. The rate of change of frequency is then defined as $\alpha = \frac{B}{T_P}$.

The transmitted waveform travels to the target at distance R and the back scattered signal from an atmospheric target, usually referred to as an echo, will be delayed by $\tau = \frac{2R}{c}$ (where c is the speed of light). The back scattered signal will also be attenuated and possibly Doppler-shifted in case the target is not stationary. The process of generating the beat frequency from the return signal can be visualized as shown in Figure 2.3.

Given that the transmitted signal frequency $f_T(t) = F_1 + \alpha t \quad \forall 0 < t < T_P$ and assuming an ideal point target, the received signal frequency f_R is given by $f_R(t) = F_1 + \alpha(t - \tau) \quad \forall \tau < t < T_P$. Mixing these two signals produces sum and difference

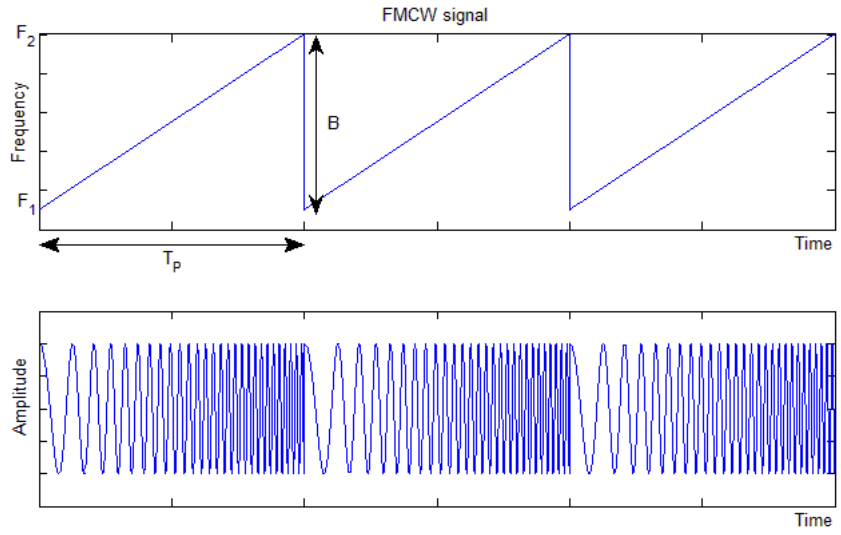


Figure 2.2: Waveform of sent signal

frequencies, $f_T + f_R$ and $f_T - f_R$. The resulting signal is then low pass filtered to remove the $f_T + f_R$ term. The term that remains is the beat frequency $f_b = \frac{B}{T_P}t$. The beat frequency term is directly related to the range by $f_b = f_T - f_R = \frac{2RB}{cT_P}$, i.e. for stationary targets the range R can be expressed as:

$$R = \frac{cT_P}{2B} f_b \tag{2.3}$$

The sampling frequency (f_{Sample}) of the A/D conversion determines the maximum beat frequency that can be detected without aliasing. Using 2.3 the maximum beat frequency determines the maximum range for FMCW radars as:

$$R_{max} = \frac{cT_P}{2B} f_{b_{MAX}} = \frac{cT_P}{2B} \frac{f_{Sample}}{2} \tag{2.4}$$

With the beat frequency and the radar parameters B and T_P known, we can retrieve range information from the return signal. The Fast Fourier Transform (FFT) is the mathematical tool used to interpret the spectrum of the beat frequency signal in terms of radar range. The point target scenario can be extended to multiple targets

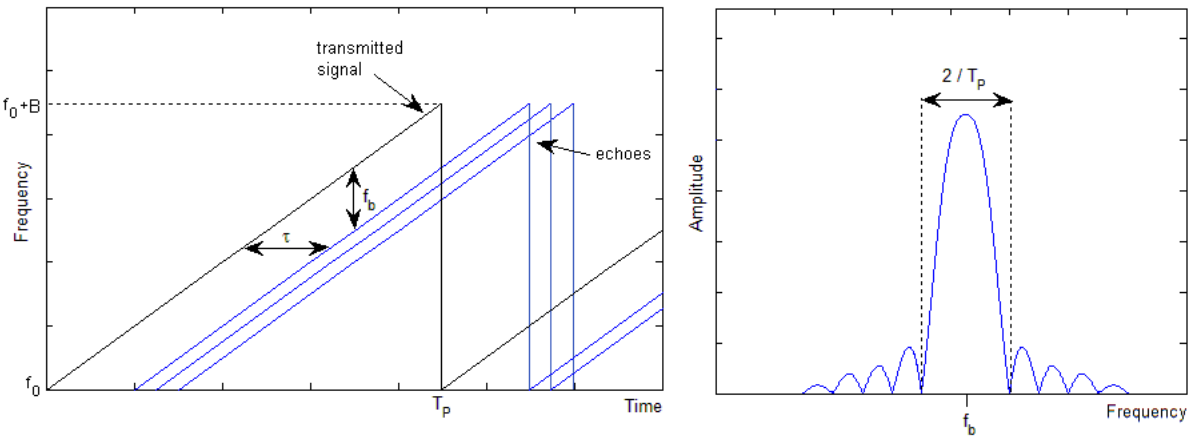


Figure 2.3: FMCW Radar Sweep (left) and Frequency diagram (right)

as well. Each target will produce a return signal with a frequency corresponding to the distance. In addition, this return signal will have an amplitude corresponding to the round trip attenuation to its specific target. These can be seen as peaks in the frequency domain.

In an FMCW radar, range resolution means the ability to separate adjacent spectra. The frequency resolution in this case is defined as $\triangle f = \frac{1}{T_P}$. The range resolution of an FMCW system is only dependent on the bandwidth of the transmitted signal. Using 2.3 the range resolution $\triangle R$ is derived as:

$$\triangle R = \frac{c}{2B} \tag{2.5}$$

Parameters such as B and T_P are configurable in an FMCW radar thus providing flexibility to adapt to the most suitable mode. For example, in order to improve range resolution the chirp bandwidth B should be increased. Keeping all other parameters constant, this will reduce the maximum range. One of the advantages of FMCW radars is that they allow for change in range resolution without change in the peak transmit power for a given sensitivity. To maximize range resolution and sensitivity, both B and T_P should be as large as possible. T_P, however, is constrained

by the coherence time of the atmospheric echo as the presented theory is based on the assumption that the target remains coherent during the sweep interval.

FMCW radars are a type of pulse-compression radar that provide exceptional range resolution when compared to pulsed radars. They operate by transmitting a long, coded waveform of duration T_P, and bandwidth B. The improvement factor they gain over pulsed radars of equivalent range resolution is given by the compression gain:

$$G_C = BT_P \tag{2.6}$$

It is possible to determine Doppler velocities by comparing the phase changes of two consecutive received pulses. In this case, the Nyquist Doppler frequency is $\frac{1}{2T_P} = \frac{PRF}{2}$ and the unambiguous velocity interval is given by

$$|v_r| \leq \frac{\lambda}{4} PRF \tag{2.7}$$

where λ is the radar wavelength, PRF is the Pulse Repetition Frequency and equals the inverse of T_P when the radar has a duty cycle of 100%. It can be shown that the response of the FMCW radar to a point target at range R_0, moving at a radial velocity v_r , can be expressed as [1]:

$$y(R) = \frac{sin\pi(f_d T_P + (R - R_0)/\triangle R)}{\pi(f_d T_P + (R - R_0)/\triangle R)} \tag{2.8}$$

where $f_d = \frac{2v_r}{\lambda}$ is the Doppler frequency. The presence of both range and Doppler terms in the argument of the sinc function illustrates the effect of target motion on the radar's ability to locate. By rearranging the terms in the argument in 2.8, the apparent range of the target R_{app} can be expressed as:

$$R_{app} = R_0 - f_d T_P \triangle R \tag{2.9}$$

where it is obvious that misregistration by one resolution cell occurs when the product $f_d T_P$ equals unity, or when $v_r = \pm\frac{\lambda}{2}\frac{1}{T_P} = \pm\frac{\lambda}{2}PRF$. Thus, targets with unambiguously measured velocities are misregistered by no more than one half a resolution cell.

Averaging is a powerful tool for increasing radar sensitivity. The minimum detectable C_n^2 can be increased by coherent and non-coherent averaging over a period of time. The number of pulses available for coherent integration is closely related to the coherence time of the observed target. It is assumed that during this coherence time, the phase of the target changes much less than a wavelength. If N_{COH} is the number of pulses available for coherent integration, the improvement in SNR due to coherent averaging is:

$$I\,[dB] = 10log(N_{COH}) \tag{2.10}$$

The choice for non-coherently averaged pulses is dictated by the amount of spatial and temporal detail needed in the observed phenomena and the time over which these phenomena don't change significantly over the spatial extent of the resolution volume. The improvement in SNR for non-coherent averaging is:

$$I\,[dB] = 10log(\sqrt{N_{NCOH}}) \tag{2.11}$$

where N_{NCOH} is the number of non-coherently averaged pulses.

2.4 Interrupted FMCW

A major disadvantage of FMCW systems is the need to use two separate, well shielded antennas for transmission and reception to prevent direct leakage of transmitted energy from overloading the receiver. Several schemes have been devised to enable FMCW radars to operate with a single antenna. In one such scheme the transmitter, antenna and receiver are connected by means of a circulator which allows energy to pass from the transmitter to the antenna and from the antenna to the receiver but isolates the receiver from direct leakage from the transmitter. Alternatively a transmit/receive (T/R) switch can be used to switch a single antenna between the transmitter and receiver. The requirement in this case is good isolation between the transmit and receiveports of the T/R switch.

Even though this system is no longer a true continuous wave radar, the essential properties of the FMCW waveform are preserved with little degradation in performance, provided suitable constraints on the switching waveform are met. This system is referred to as a frequency modulated interrupted continuous wave (FMICW) radar. FMICW systems have the advantage of increasing the portability of the radar system, simpler antenna alignment geometry, and relaxing the isolation requirements between transmitter and receiver, but have the disadvantage of limited range due to the switching constraints.

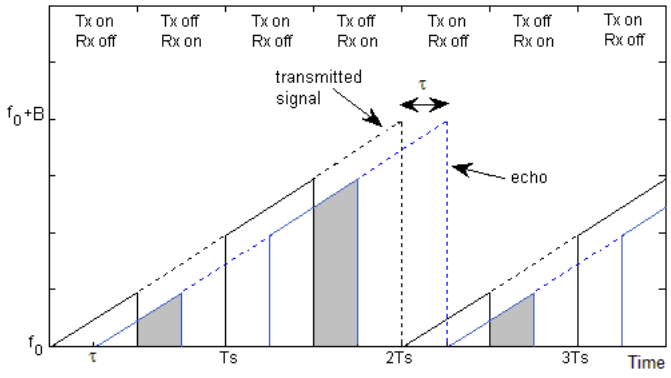


Figure 2.4: FMICW principles

A scheme is possible in which a 50% duty cycle square wave is used as the T/R switching waveform [6]. The transmitter and receiver may be connected to a single antenna by a T/R switch. This scheme has the advantage of maximising average transmitted power. The effect of switching with a 50% duty cycle is depicted in Figure 2.4.

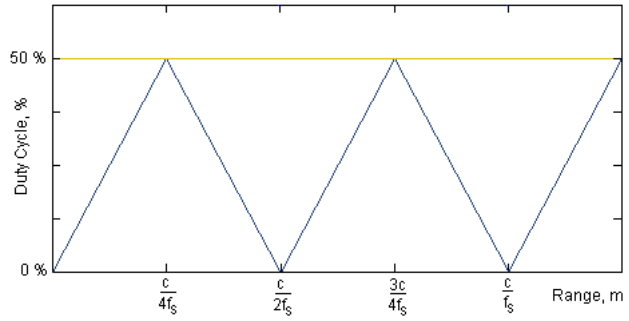


Figure 2.5: FMICW Sensitivity function

The FMICW receiver output signal is the unswitched FMCW beat signal multiplied by a pulse train with switch frequency f_S, and duty cycle which depends on target range as shown in Figure 2.5. It can be seen that when the duty cycle is 50% the echo delay equals half the switch period. The switch frequency f_S is chosen to give the maximum sensitivity at maximum range (R_{MAX}), thus:

$$f_S = \frac{c}{4R_{MAX}} \tag{2.12}$$

As it is shown in Figure 2.5, there exist some ranges whose associated sensitivity is null. These ranges are called blind ranges:

$$R_B = \frac{c}{2f_S} n \quad n = 0, 1, 2... \tag{2.13}$$

The received signal is an echo from the target modulated with a rectangular wave. This modulation produces copies of the spectrum shifted $+nf_S\,Hz$, where n is an integer. This introduces the condition that the switching frequency must be larger

than twice the highest beat frequency $f_{b,MAX}$ to avoid spectral aliasing.

$$f_S > 2f_{b,MAX} = f_{Sample} \tag{2.14}$$

Then, using that and equations 2.4 and 2.12, it follows that:

$$R_{MAX} < \sqrt{\frac{c^2 T_P}{16B}} \tag{2.15}$$

CHAPTER 3

SYSTEM DESCRIPTION

The current FMCW radar project started as an analog of the S-Band FMCW boundary layer profiler developed in 2003 at the University of Massachusetts at Amherst [1]. The change from S-Band to UHF was proposed in order to reduce the Rayleigh scattering from insects and birds which appeared to dominate the observed vertical profile of the mean reflectivity at S-Band. The current FMCW Wind Profiler was started in the summer 2006 by the former students Albert Genis during 2008 [7], Iva Kostadinova in 2008-2009 [8], and David Garrido [2, 3] in 2009. This chapter reviews the current design of the Wind Profiler system.

3.1 Current Hardware Configuration

The Wind Profiler is an FMCW radar that operates in the 900 MHz ISM frequency band (902-928MHz) with a center frequency of 915MHz. The chirp is generated by a Direct Digital Synthesizer (DDS) with a bandwidth up to 25MHz and linear FM modulation. It is able to achieve a maximum range resolution of 6m. Table 3.1 lists the basic system specifications and Figure 3.1 depicts the major blocks of the current Wind Profiler design.

The default mode of operation had a PRF of 100Hz, and a duty cycle of 100% (T_P =10ms). Using Equation 2.7 it is possible to calculate the vertical unambiguous velocity up to 8.2 m/s which exceeds the range of usual turbulent velocities on the boundary layer (3-5m/s). All the signals needed to start the radar, such as those used to start and set up the DDS are generated by a Field Programmable Gate Array Board (FPGA). The parameters of the radar such as bandwidth, PRF and T_P are configurable. This allows for adaptation to the resolution and scenario as necessary.

The antennas are 4ft diameter parabolic dish antennas with dipole antenna feeds,

Transmitter	
Center Frequency	915 MHz
Peak transmit power	30 W (44.77 dBm)
Transmitter type	Solid State RF
Bandwidth	$\leq$25 MHz
PRF (100% duty cycle)	100 Hz
Antennas	
Type	Four Parabolish dish
Gain	18 dB
Polarization	Linear
Front to Back Ratio	22 dB

Table 3.1: System specifications

19^{o} beamwidth and 18dB of gain. The antennas have a broad beam, in order to employ a SA technique for estimating horizontal winds.

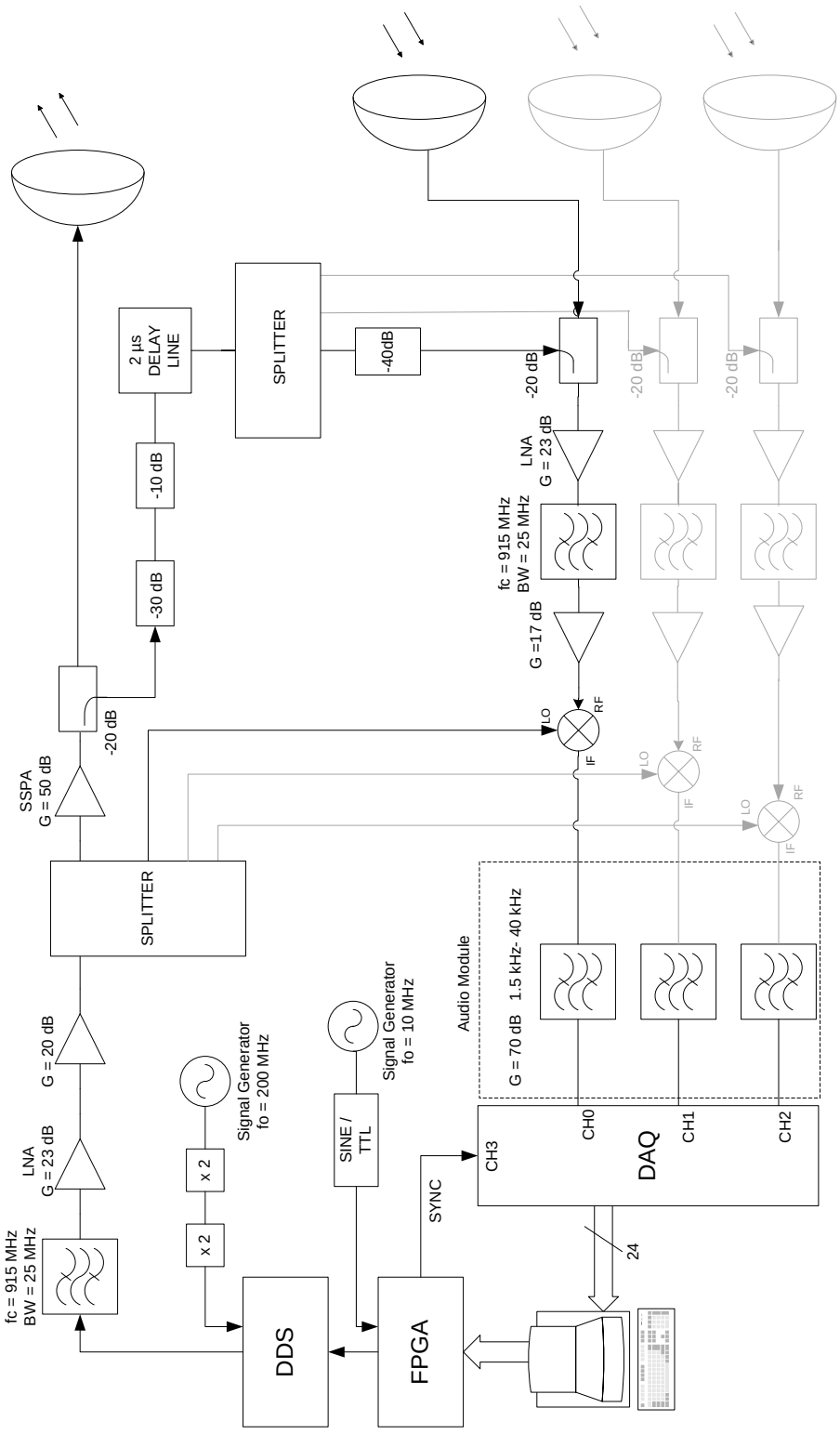


Figure 3.1: Current FMCW Wind Profiler block diagram

3.2 Windprofiler Subsystems

Figure 3.1 shows a simplified block diagram of the FMCW Wind Profiler. Currently all the hardware, apart from the computer which contains the Data Acquisition Card (DAQ) and the audio module with its power supply, is in 4 boxes (FPGA Box, Big Amplifier Box, RF Box and Power Supply Box). The current state of the boxes as well as the connections between them can be found in Appendix C.

3.2.1 Control and Transmit Subsystem

The FPGA (Altera Cyclone II EP2C20) generates all the radar signals required for the proper operation of the Wind Profiler . It uses a clock at 10MHz. The design is characterized by the generation of all signals in total synchronization, giving no room for timing skews that can lead to phase wrapping in the received data. Besides, the duty cycle of the Wind Profiler is set up to 100%. The operator stores the necessary parameters to configure the radar in a file which will be sent via serial interface to the FPGA. That file and its parameters can be modified at any time, making the radar flexible to operate in different contexts (see Figure 3.2).

The transmitted chirp is generated by a 10-bit Digital Signal Synthesizer from Analog Devices (AD9858 DDS). The DDS receives the needed parameters for frequency sweep generation from the FPGA. The DDS board uses an external 800MHz clock signal that comes from a 200MHz oscillator and two doublers. The analog mixer mixes the clock signal with the chirp centered at 115 MHz to produce the desired product at 915 MHz. The undesired mixing products are filtered and the signal is amplified. A 4-way splitter splits the transmitted signal to the three LO inputs of the receiver mixers and to the RF high power amplifier. The high power amplifier is a linear power solid state RF amplifier from OPHIRRF with a gain of 50dB providing a 30W (44.8 dBm) output. A 20dB directional coupler delivers the output of the amplifier to the antenna and couples part of it to the calibration loop.

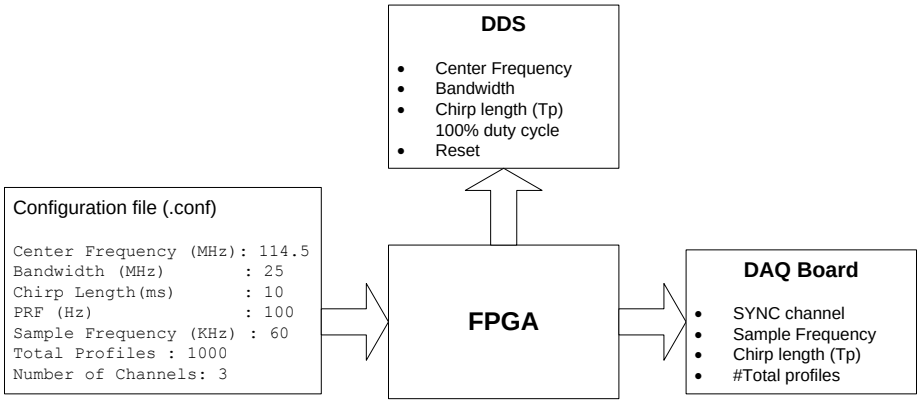


Figure 3.2: FPGA Control signals generation

3.2.2 Calibration Loop

The calibration loop allows continuous monitoring of the system performance and gives a reference point for the evaluation of the radar sensitivity. The Wind Profiler uses a bulk acoustic wave (BAW) delay line, with an effective delay of 2μs from Teledyne Electronic Devices. An attenuated replica of the transmitted signal serves as its input. This delay will produce an echo corresponding to a fixed range of 300m. The output obtained is coupled into the receiver chain through a 20dB coupler. The delayed chirp produces a signal of a fixed f_b at the mixer's IF port. Using the default configuration (T_P=10ms, B=25MHz) the beat frequency is 5kHz (see equation 2.3). Other fixed beat frequencies will appear at $3f_b$ and $5f_b$ (corresponding to 900m and 1500m) due to the triple and quintuple travel signal in the delay line.

3.2.3 Receiver Subsystem

The receiver chain consists of three channels for the three receiving antennas which would allow the SA technique to retrieve horizontal winds. The calibration signal is coupled to the received signal in the front end. The coupler is a 20dB coupler from Narda Microwave East. To ensure a good noise figure, the first component in the receiver is a 23dB low noise amplifier. The next components in the chain are a

ceramic band pass filter from Lark Engineering and a 17dB amplifier.

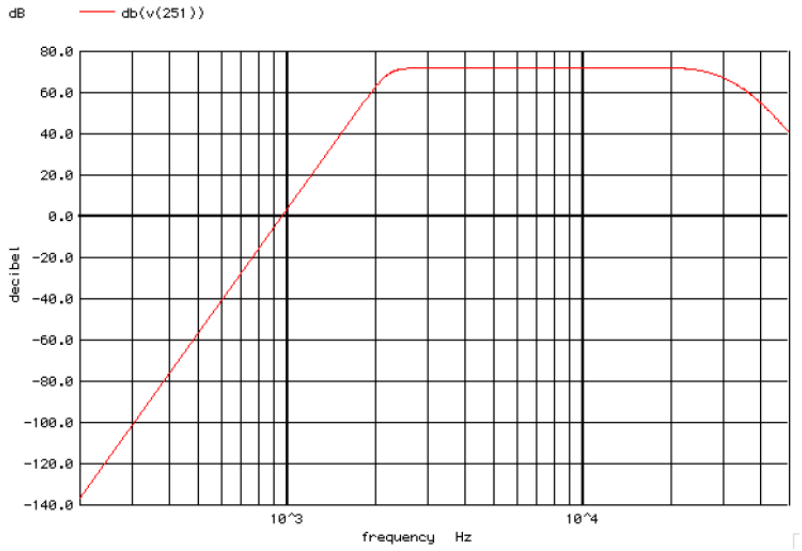


Figure 3.3: Audio module filter response

The output of the amplifier is passed on to the RF port of the mixer. Then the return signal is mixed with an attenuated copy of the transmitted signal through a high power mixer from Mini Circuits (ZEM-4300MH+). The IF signal at the mixer output is fed into the audio module [2] designed with low noise components. The audio module filter response can be seen in Figure 3.3. Finally, the amplified base band signal is digitalized by a four channel Analog-Digital Converter (ADC) and stored on a local computer.

CHAPTER 4

DATA ACQUISITION

A new Data Acquisition Board was integrated in the radar. The board is a four-channel low-noise 24-bit delta sigma PC104-Plus from General Standards. Data acquisition programs were developed for it, but there were synchronization issues that should be corrected. The adopted solution and all relevant information will be showed in the following sections.

4.1 Acquisition Board

Figure 4.1: 24DSI6LN Data Acquisition Card

The Data Acquisition Card in the current Wind Profiler is a 24-bit 24DSI6LN from General Standards. The board has 4 input channels and it allows sample rates from 2 KSPS to 200 KSPS per channel with an accuracy of 25 PPM. Each of four analog input channels contains a low-pass image filter and a delta-sigma A/D converter that

provides digital antialias filtering. A linear-phase digital antialiasing filter rejects out-of-band signals, and a low-pass analog filter reject those interference signals that fall within the harmonic images of the digital filter.

The configured voltage range is ±10V corresponding to a maximum input power of 30dBm for a single tone and the dynamic range of the board is 110dB. All the specifications can be found in [9]. All data transfers are long-word D32 (32 bits). The width of the data field is adjustable from 16 bits to 24 bits, in this case it is set to 24 bits. Each value in the data buffer consists of a 5-bit channel tag field and a data field, as shown in Table 4.1 [10].

DATA WIDTH	RESERVED (Zero)	CHANNEL TAG	CHANNEL DATA VALUE
24 bits	D[31..29]	D[28..24]	D[23..0]

Table 4.1: Input Data Buffer Organization

Input data values are configured to be represented in offset binary format instead of two's complement format. The Full-scale Range (FSR) is 20V (±10V). One LSB equals the full-scale range divided by 2^{24}:

$$1LSB = \frac{20}{2^{24}} = 1.192\mu V \ = 10\log\frac{(1.192\mu V)^2}{50} + 30 = -105dBm \qquad (4.1)$$

The Delta Sigma architecture is based on the technique of oversampling to reduce the noise in the band of interest, which also avoids the use of high-precision circuits for the antialiasing filter. In Delta Sigma converters, noise is further reduced at low frequencies, which is the band where the signal of interest is, and it is increased at the higher frequencies, where it can be altered. Due to the oversampling the external input clock needs to be very fast (25.6-51.2 MHz [9]). With the old DAQ board it was possible to use a clock provided by the FPGA, achieving a totally synchronized system. At the moment, this is not possible because the fastest clock speed the FPGA can provide is 10MHz.

For the signal processing, it is necessary to determine both the beginning and

the end of each profile. Therefore, some kind of synchronization is needed between the DAQ and the FPGA. One possible solution could be to use an external sync signal with a period of T_P, and then, to acquire certain amount of data (which would correspond to a profile) at every positive edge of the sync signal. This type operation is known as bursting (capturing a fixed number of samples at a time). In this case, it is not possible because this board doesn't support any type of bursting feature. The chosen solution consists of using the signal at frequency of $\frac{1}{T_P}$ provided by the FPGA as input for the analog channel 3 (because the board has 4 channels and only 3 are needed for the 3 receiver antennas), and adapting the acquisition program to store the corresponding number of samples at each positive edge of the sync signal. System cable signal pin assignments are described below (for further information refer to [10]).

ROW-A	
PIN	FUNCTION
3	INP CHAN 00 LO
4	INP CHAN 00 HI
7	INP CHAN 01 LO
8	INP CHAN 01 HI
11	INP CHAN 02 LO
12	INP CHAN 02 HI
15	INP CHAN 03 LO
16	INP CHAN 03 HI

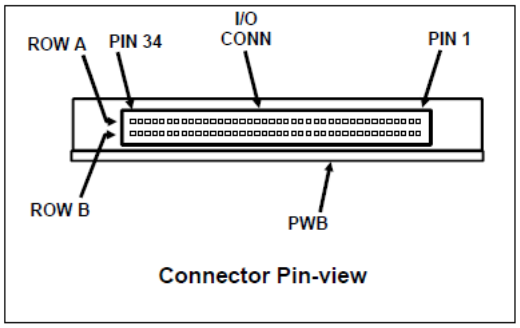


Figure 4.2: DAQ Connector

4.2 Data acquisition program

A new program has been developed to collect data. It can be found in `/usr/src/kernels/.../drivers/24dsi/savedata_windprof`. Table 4.3 lists all the files contained in this folder and describes them briefly. This program is initially based on the "Save Acquired Data sample" application provided by General Standards Corporation [11]. The source files are provided and explained in Appendix A. The main program configures the board according to the desired parameters, and acquires the data from channels 0-2. As explained before, it is necessary to synchronize the collected data to determine when a profile starts and ends. For this purpose, the program also acquires channel 3 which corresponds to the synchronization signal at 100Hz ($\frac{1}{T_P}$).

File name	Description
`main.c`	The main program. Contains all the procedures needed to configure the board and collect the data.
`savedata.c`	Contains some useful functions. Currently only it is used by the main application the functions `_channels_profiler`, `_save_data` and `dsi_config_board_profiler`.
`fmcw_conf.c`	Contains `read_fmcw_conf` and `fwrite_fmcw_conf` functions to read and write the `.conf` file which is also used by the FPGA.

Table 4.3: Files in `savedata_windprof/`

Before executing the program it is necessary to start the driver manually. Log in as root user, change to the directory where the drivers are installed, which may be `/usr/src/kernels/.../drivers/24dsi/driver`. Then, install the driver module and create the device nodes by executing the command `./start`. It is possible to verify that the device driver module has been loaded by issuing the command `lsmod` and examining the output (the module name 24dsi should be included in the output) . To collect data, the program needs to run while the radar is operating, i.e. the programs that start the radar have to be executed beforehand. To execute the acquisition program, change to the directory where the acquisition application is installed (`/usr/src/kernels/... /drivers/24dsi/savedata_windprof`), build the applica-

tion by issuing the command `make all`, and start the application issuing the command `./savedata -m`, where the parameter `m` is optional, which represents the maximum duration in minutes of the data acquisition.

The main program works as follows. First the board is identified and selected, the device is opened, and the board is configured using FMCW defaults. The file in which the data will be stored is created, has this name format: ``#month#day#hour#minute.dat'' and it will be stored in `windprofiler2010/data/`. The `.conf` file is read and the appropriate header is written. There are some necessary parameters defined in the `.conf` file: the sample frequency f_{Sample} , the total number of profiles and the chirp length T_P . The acquisition starts. First of all it reads only channel 3, which contains the sync signal, looking for its first positive edge. To find the edge a value for the threshold is set approximately to 1.65V because the sync signal generated by the FPGA is a square wave with 3.3V of amplitude. The Block Size is the number of scans per profile. When a positive edge is found, Block Size samples of the data channels are saved. The Block Size is calculated as $f_{Sample} \cdot T_P - 5$ instead of $f_{Sample} \cdot T_P$ so there is enough time to look for the next edge. The program will be looking for the next positive edge of the sync signal, and will save the next block until the number of profiles read in the `.conf` file is reached or a timeout occurs.

CHAPTER 5

SIGNAL PROCESSING

New signal processing programs have been developed. The previously programs were too slow and they needed to be improved as well since they used some kind of preprocessing so as to find out the beginning and the end of the profiles. This it is not needed anymore thanks to the new acquisition program. Signal real processing program has also been implemented. All these programs will allow the radar to be tested in an easier way both in the laboratory, and later, in the field deployment.

5.1 Signal processing programs

After the data acquisition, it is necessary to process it properly to extract the required information. In the Wind Profiler, two different kinds of information can be extracted from the data: range and doppler velocity. To perform the data processing, programs in IDL [12] have been developed. In this section these programs are described and all the source code is provided in Appendix B.

The program called `windprof_process` reads the raw data from the file created by the acquisition program with a `.dat` extension, and it will create another file with the same name, but with a `.fig` extension. This file will contain the averaged power, the Doppler velocity, the matrix mean and correlation coefficient for each set of the averaged profiles.

The program requires the name of the raw data file with the `.dat` extension, the time for noncoherent integration and the number of the channel are required as input of the program. Apart from this, there are other options which serve as input:

- `mr:` If `mr=1`, means removal is performed, and ground clutter rejection and rejection of stationary targets is achieved. `mr=0` will be useful for testing when it is required that a calibration signal appears.

- `pc`: If `pc=1` a phase correction is carried out.
- `delete_samples`: The number of samples to delete at the beginning of each chirp. As shown in Figure 5.1 the chirp has a transient which contains no information and these samples can be deleted.
- `png`: If `png=1` the plot is saved in a .png file (it can be found in `windprofiler2010/data/`).

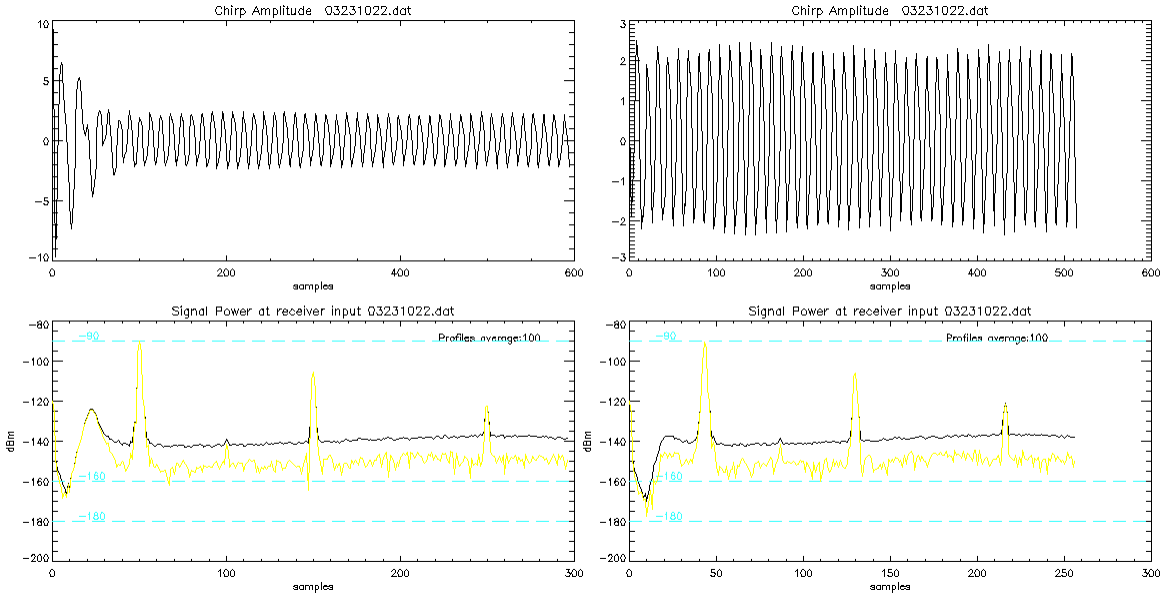


Figure 5.1: Amplitude of a chirp and Power Spectrum

The program's structure is as follows. It reads the header of the raw data file and writes the corresponding header to the output file. It reads the number of profiles available for noncoherent integration (`prof_avg`) and stores it in a matrix (`f_frame`). The first 8 bits of each sample are discarded because they indicate the channel number, and by using 4.1 the sample's value is transformed to real voltage value. A Fast Fourier Transform is done on the matrix. A phase correction is needed at this stage, which will be explained in section 6.1 Synchronization. The mean of the FFT from each sweep is calculated, if `rm=1` the mean from the FFT is removed. The matrix is multiplied by its conjugate to compute the average power. The pulse-pair technique described in Section 2.3 is used to estimate the mean Doppler velocity. Then, it calculates the

autocorrelation of the matrix with lag of one sweep and computes Doppler velocity from the phase of the autocorrelation. The correlation coefficient is also calculated. All this process is repeated until the end of file is reached. Finally all this information for each set of averaged profiles is stored in the output file with the `.fig` extension.

The IDL procedure called `windprof_display` reads the `.fig` file and after some range correction, it creates 2D images of the radar reflectivity and Doppler velocity.

5.2 GUI

The procedure to use the FMCW Radar and the main programs used are summarized below (for further details see Appendix B):

1. Start DAQ drivers	`.../24dsi/driver/start.c`
2. Start Radar	`fmcw_current.conf, windprofiler_0409.c`
3. Start acquisition	`.../savedata_windprof/savedata.c`
4. Processing of raw data	`windprof_process.pro`
5. Display data	`windprof_display.pro`

Table 5.1: FMCW Radar collection&display data procedure

A graphical user interface (GUI) has been created with the purpose of managing all the operations and making the operation of the Wind Profiler easier. To start the FMCW Radar GUI, go to the `windprofiler2010/` directory and execute the command `idl wgui`. The GUI interface is shown in Figure 5.2.

When the button "Start Radar" is pressed the parameters introduced in Radar and Acquisition Settings are stored in the `fmcw_current.conf`. Then, the `windprofiler_0409.c` is executed, and it sends the parameters to the FPGA through the serial port, and initiates the Radar. If "radar reset" button is pressed the FPGA is reset and the Wind Profiler stops. It is important to bear in mind that the acquisition program also uses the `.conf` file, so these parameters must not change between radar initialization and data acquisition. The "Start Acquisition" and "Stop Acquisition" buttons execute and kill the data collection program respectively.

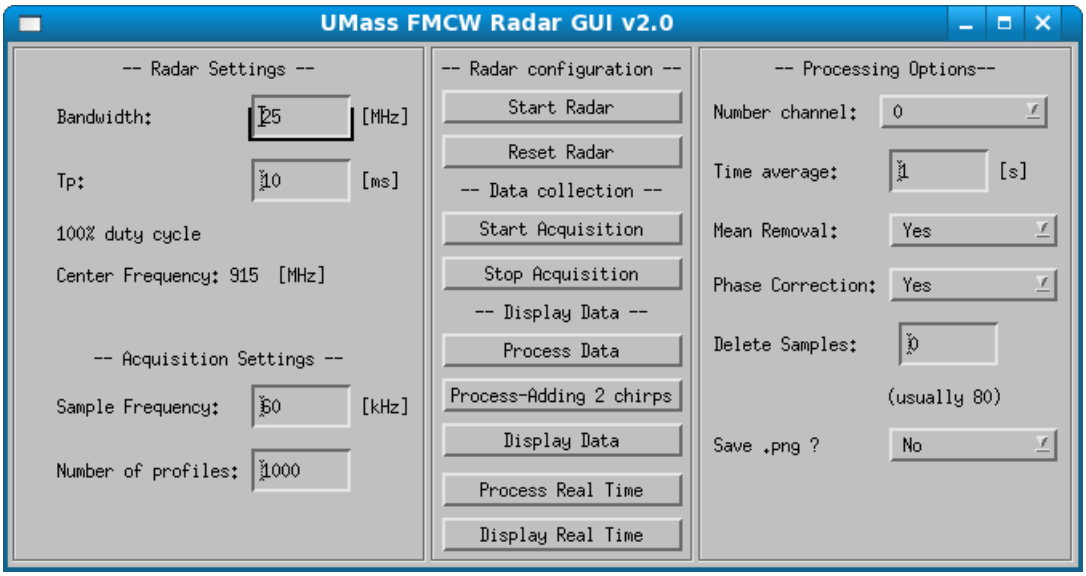


Figure 5.2: FMCW radar GUI

When either "Process Data" or "Display Data" is pressed, a window is opened where the user can pick the file to process and display respectively. The processing program (`windprof_process.pro`) uses all the parameters shown to the right side of the GUI.

A Real Time processing program also has been developed. It is possible to process and display data while acquisition is running. As soon as the acquisition has started, "Process Real Time" must be pressed, followed by "Display Real Time".

CHAPTER 6

RADAR TESTS

Following the changes to the acquisition program and the implementation of the new signal processing programs outlined in the previous chapter, a series of laboratory tests was done to evaluate the system performance. In these preliminary tests, the radar operates with only one receiving antenna and it is able to analyze scattering in the vertical direction. Their setup and the results are described here.

6.1 Synchronization

Once the new data acquisition program finishes, it then proceeds to check the timing, a correct synchronization will allow the computation of the doppler information. The results are shown on Figure 6.1. This figure corresponds to when the FFT of the averaging of 100 profiles and mean removal is carried out. It shows the power spectrum of the averaged profiles (left) and the phase of the bin which is equivalent to 300m (right). The phase of the 400m bin also is plotted with the purpose of comparing it with the 300m bin.

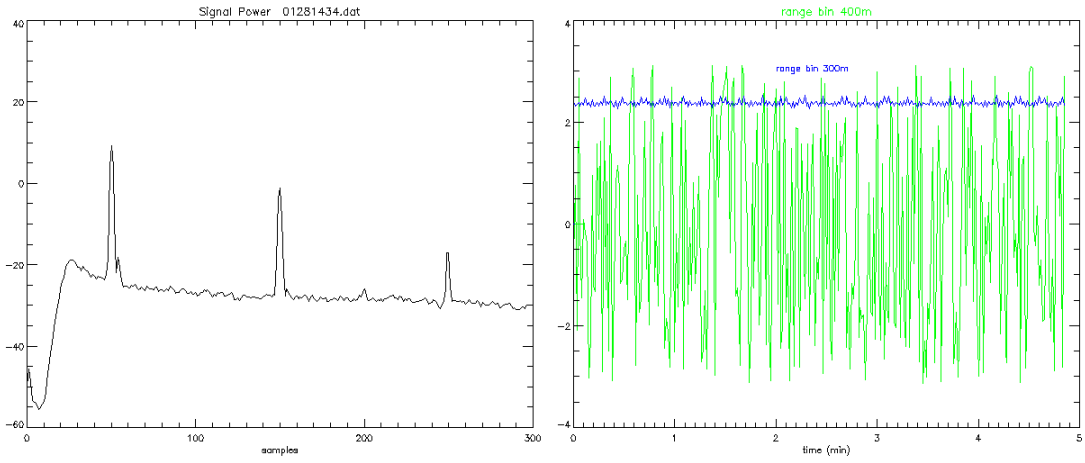


Figure 6.1: Power spectrum (left) and Phase of the 300m bin and 400m bin (right)

It is seen that the peaks corresponding to the calibration (300m, 900m and 1500m)

appears in spite of performing the mean removal to get rid of stationary targets. As expected these peaks appear because its phase is not 0. The standard deviation of these peaks without removing the mean of the FFT are $\sigma_{300m} = 0.061$, $\sigma_{900m} = 0.039$, and $\sigma_{1500m} = 0.040$. Since the FPGA clock is not the same as the DAQ clock, they are not fully synchronized and there is a jitter which translates into an uncertainty in the phase. Moreover, this is different for each frequency. To fix this, the following correction is performed:

$$\Phi(f) = \Phi^*(f_{CAL}) \cdot \frac{f}{f_{CAL}}$$

where f_{CAL} is the calibration frequency corresponding to 300m and $\Phi^*(f_{CAL})$ is the conjugated phase at f_{CAL}, thus ensuring that the phase will be 0 at range of 300m. The results with the phase correction are shown in Figure 6.2. The peaks of the calibration signal have disappeared and its phase is random. Now the standard deviations without mean removal are $\sigma_{300m} = 1.76 \cdot 10^{-18}$, $\sigma_{900m} = 0.00024$ and $\sigma_{1500m} = 0.0027$, so the uncertainty of the phase is significantly reduced.

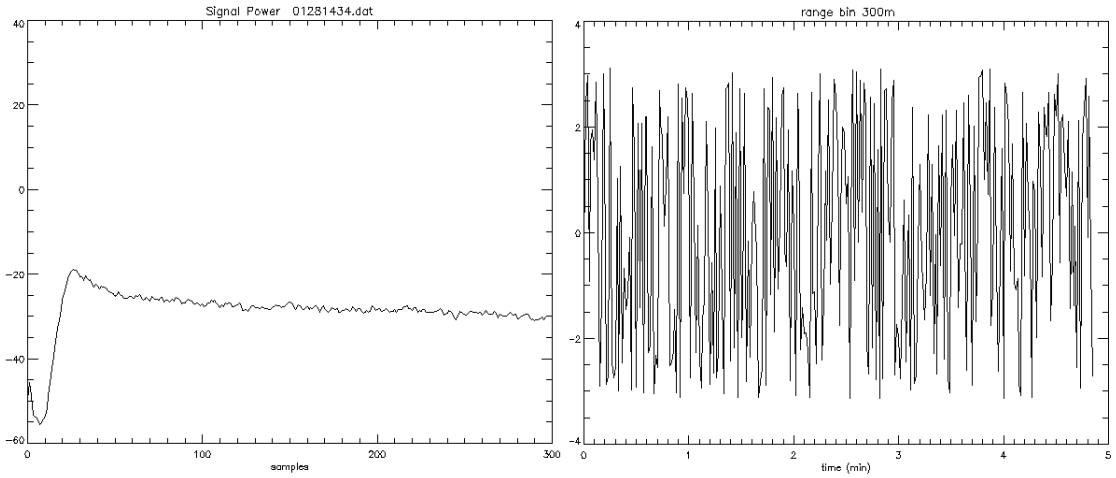


Figure 6.2: Power spectrum (left) and Phase of the 300m bin (right) with phase correction

6.2 Minimum detectable Signal

For bistatic weather radar with Gaussian shaped beams, the received mean power at the antenna port from a scattering volume in the far field is [13]:

$$P_r = \frac{P_t G_t G_r \lambda^2 \theta^2 \triangle R \, \eta}{512 \ln 2 \pi^2 R^2} C_a$$

The parameter C_a accounts for the antenna parallax existing in every bistatic radar and is given by $C_a = exp(-2 \ln 2 \frac{d^2}{\theta^2 R^2})$ where d is the distance between the centers of the antennas in meters. Using (2.1), the structure constant can be expressed in terms of the received power as :

$$C_n^2 = \frac{P_r \, 512 \, \ln 2 \pi^2 R^2}{P_t G_t G_r 0.38 \lambda^{5/3} \theta^2 \Delta R \, C_a} \tag{6.1}$$

Typical values for the structure constant C_n^2 vary from $10^{-15} m^{\frac{-2}{3}}$ for weak turbulence to $10^{-13} m^{\frac{-2}{3}}$ for intensive turbulence within the convective boundary layer. Besides, the parameter C_n^2 depends on the season and the weather. Going from $10^{-17} m^{\frac{-2}{3}}$ in a cold, dry day in winter, to $10^{-12} m^{\frac{-2}{3}}$ in a hot, and humid day. The range of the convective boundary layer is below 3Km, typically 2Km in New England. And the usual turbulent velocities can reach 3-5 m/s. Knowing the minimum received power and using equation 6.1 it is possible to find what the minimum structure constant C_n^2 is. These values are plotted in Figure 6.3 for different values of received power (P_r).

The set up to measure the noise floor is depicted in Figure 6.4. To simulate the leakage, a high power load, cable and attenuators are used. The frequency of the simulated leakage is $f_{leak} \simeq 210Hz$ ($B = 25MHz$, $T_p = 10ms$) which corresponds at range of $R_{leak} = 12.6m$ and is in good agreement with the previously obtained one in the field deployment [8].

The expected performance of the delay line was experimentally validated, and this

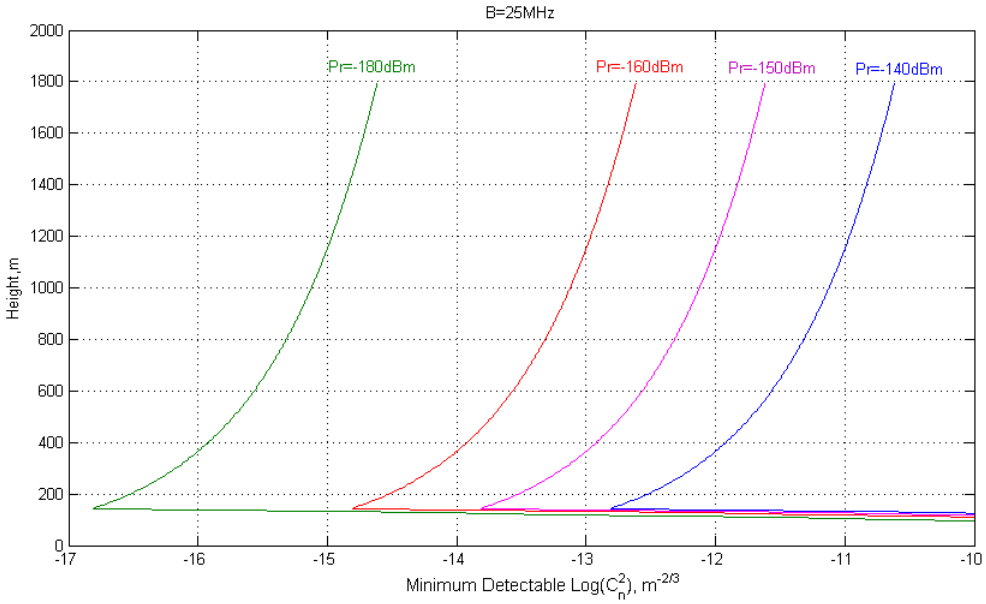


Figure 6.3: Minimum detectable with default configuration

was used to determine the true available SNR and minimum detectable signal in the Wind Profiler. The transmitted power is $44.8dBm$. The total attenuation calculated in calibration path is $L_{CAL\,Path} \simeq 135dB$. Therefore, the level of the calibration signal in the receiver input is $P_{CAL} \simeq -90dBm$. If not stated otherwise in the following outcomes the default configuration ($B = 25MHz$, $T_p = 10ms$, $PRF = 100Hz$) is used.

An amplifier of 17dB was added in the front end to improve the radar sensitivity (see Figure 6.5). The disadvantage of this increasing of the sensitivity is that the maximum leakage allowed decreases. As stated previously, using the current configuration: $P_{leak\,MAX} \simeq -50$ dBm.

At this point if another amplifier is added in the front end, the noise floor is increased by its gain. Thereby the noise figure in a receiver is primarily determined by the first components in the receiver chain.

As shown in Figure 6.5 the noise floor when the 17dB amplification is added is around -140dBm, thus the minimum detectable C_n^2 in that case is between $10^{-12.5}m^{\frac{-2}{3}}$

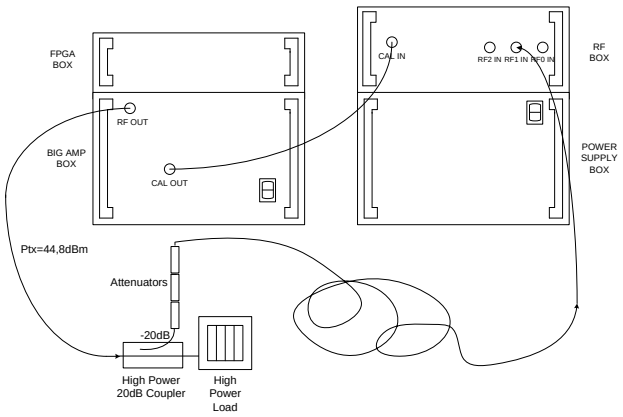


Figure 6.4: Lab Setup

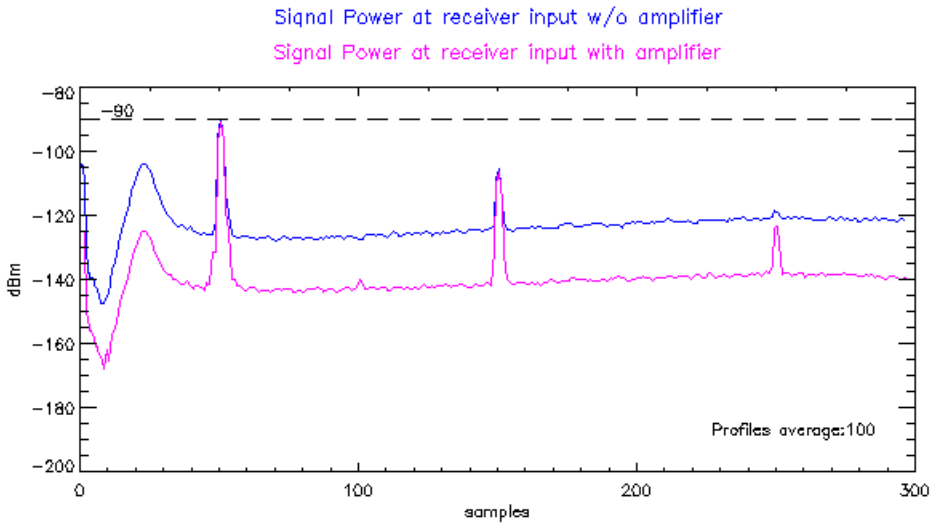


Figure 6.5: Noise floor with and without 17dB amplifier

and $10^{-10.5}m^{\frac{-2}{3}}$. It is seen that the radar can only detect very high values of the structure constant which correspond to high intensity refractive index fluctuations due to very strong turbulence. There are some options to improve the sensitivity in the signal processing. The non-coherent averaging will improve the SNR and smaller structure constant values will be able to be detected, reducing the temporal resolution of the Wind Profiler. It is possible to have an improvement of 15dB (see Equation 2.11) with averaging of 1000 profiles, i.e. 10sec. The result is plotted in the next figure (Figure 6.6).

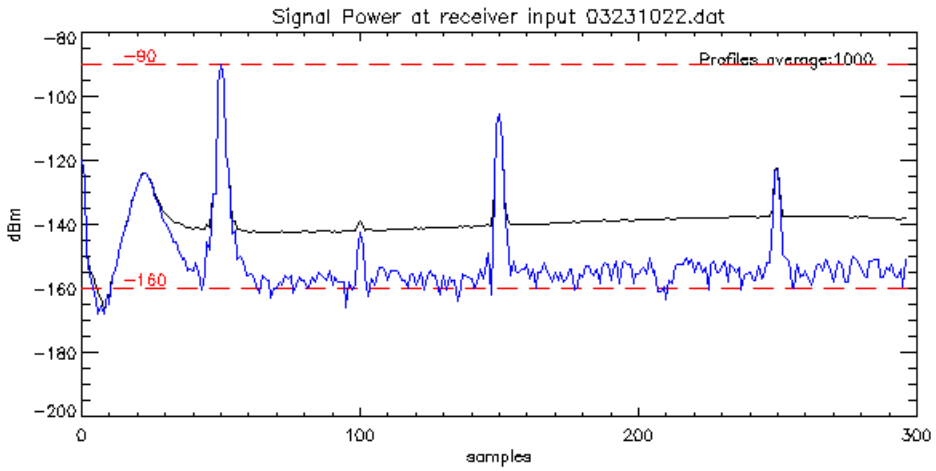


Figure 6.6: Noise floor with averaging of 1000 profiles

In the same way, it is possible to improve 6dB more by adding 2 consecutive chirps. This will reduce the effective PRF and therefore the maximum unambiguous velocity (see Equation 2.7).

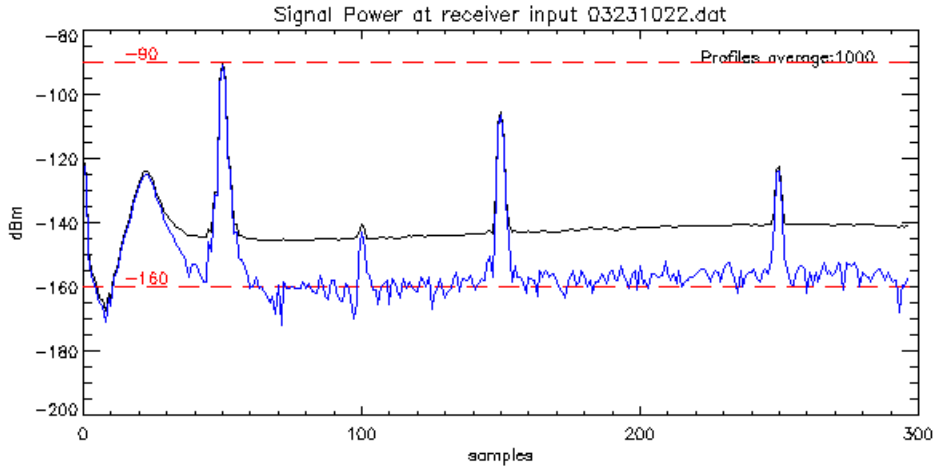


Figure 6.7: Noise floor adding 2 profiles

If the noise floor is below -160dBm it is possible to measure values of the structure constant C_n^2 up to $10^{-14}m^{\frac{-2}{3}}$ depending on the bandwidth and the height. This is depicted in Figure 6.8.

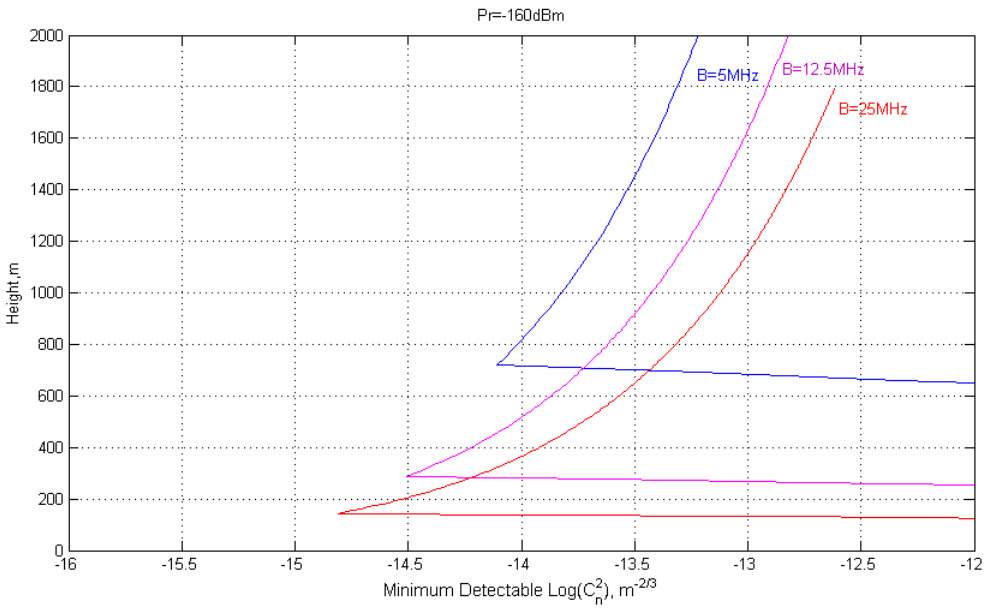


Figure 6.8: Minimum detectable C_n^2 when received power is -160dBm

By varying the sweep time (T_p) and the bandwidth (B), the radar can be operated in a mode suitable for particular atmospheric conditions. For example, to improve range resolution the chirp bandwidth B should be increased. Keeping all other parameters the same, this will reduce the R_{max}. The maximum unambiguous Doppler velocity ($v_{r,\,max}$) in this case will remain the same. If higher unambiguous velocities are desired, T_p should be decreased keeping B the same. This will not affect the range resolution but it will again decrease the maximum range. One of the advantages of FMCW radars is that they allow for change in range resolution without change in the peak transmit power for a given sensitivity.

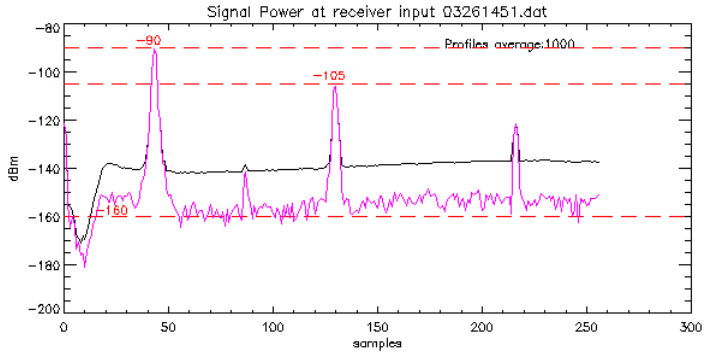
A trade-off exists between the parameters of the windprofiler, mainly ΔR, R_{max} and $v_{r,\,max}$. The minimum Range (R_{MIN}) must also be included due to the cut frequency of the audio module at $2.17kHz$. In the following table (Table 6.1) these parameters are calculated for different configurations (values of T_P, B and f_{Sample}) and in Figure 6.9 some results are plotted.

T_P	**10 ms**	**10 ms**	**10 ms**	**5 ms**	**5 ms**	**5 ms**	**20 ms**
B	**25 MHz**	**12.5 MHz**	**5 MHz**	**25 MHz**	**25 MHz**	**12.5 MHz**	**25 MHz**
f_{Sample}	**60 kHz**	**60 kHz**	**60 kHz**	**60 kHz**	**120 kHz**	**60 kHz**	**60 kHz**
ΔR	6 m	12 m	30 m	6 m	6 m	12 m	6 m
R_{max}	1800 m	3600 m	9000 m	900 m	1800 m	1800 m	3600 m
$v_{r,\,max}$	8.2 m/s	8.2 m/s	8.2 m/s	16.4 m/s	16.4 m/s	16.4 m/s	4.1 m/s
$R_{CAL} = 300m$							
$f_{b,CAL}$	5 kHz	2.5 kHz	1 kHz	10 kHz	10 kHz	5 kHz	2.5 kHz
$f_{c,filter} \simeq 2.5kHz$							
R_{min}	150 m	300 m	750 m	75 m	75 m	150 m	300 m
$R_{leak,lab} = 12.6m$							
f_{Leak}	210 Hz	105 Hz	42 Hz	420 Hz	420 Hz	210 Hz	105 Hz

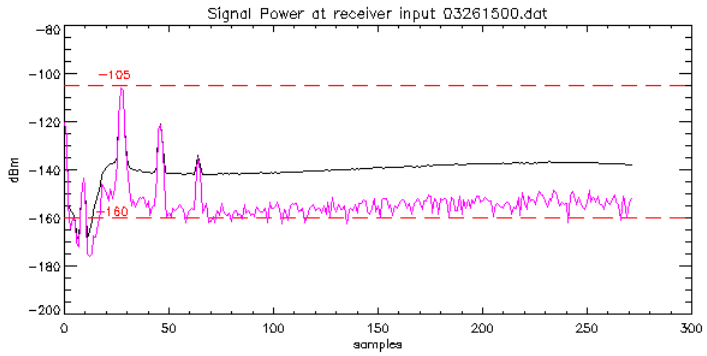
Table 6.1: FMCW parameters

As it is shown on Figure 6.9.a the dominant $f_{b,CAL} = 5kHz$ frequency is seen along with the triple travel chirp at 15 kHz and the quintuple travel chirp at 25kHz which are about 15dB and 30dB respectively below the main 5kHz signal, as expected. In Figure 6.9.b and Figure 6.9.c it is seen the effect of the cut-off frequencies of the audio module filter ($f_{c,low} = 2.17kHz$ and $f_{c,high} = 29.3kHz$).

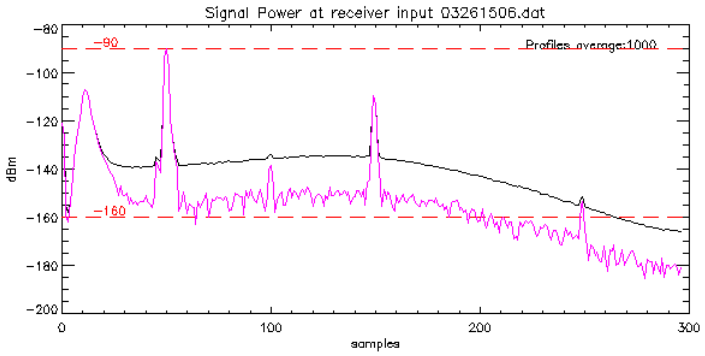
(a) Default:$B = 25MHz$, $T_P = 10ms$, $f_{Sample} = 60kHz$



(b) $B = 5MHz$, $T_P = 10ms$, $f_{Sample} = 60kHz$



(c) $B = 25MHz$, $T_P = 5ms$, $f_{Sample} = 120kHz$



(d) $B = 25MHz$, $T_P = 20ms$, $f_{Sample} = 60kHz$

Figure 6.9: Noise floor with different configurations (Averaging of 1000 profiles)

The detectable minimum also is calculated with a simulated leakage of -40.5dBm. To prevent saturating the audio module an attenuator of 10dB is added in its input. The result is plotted in the next figure (Figure 6.10), using default configuration, 10sec of averaging and adding 2 chirps. It is seen with these conditions the noise floor is around -150dBm, thus strong turbulence could be measured with values of the structure constant C_n^2 between $10^{-13}m^{\frac{-2}{3}}$ and $10^{-12}m^{\frac{-2}{3}}$, and velocities up to 4.1 m/s.

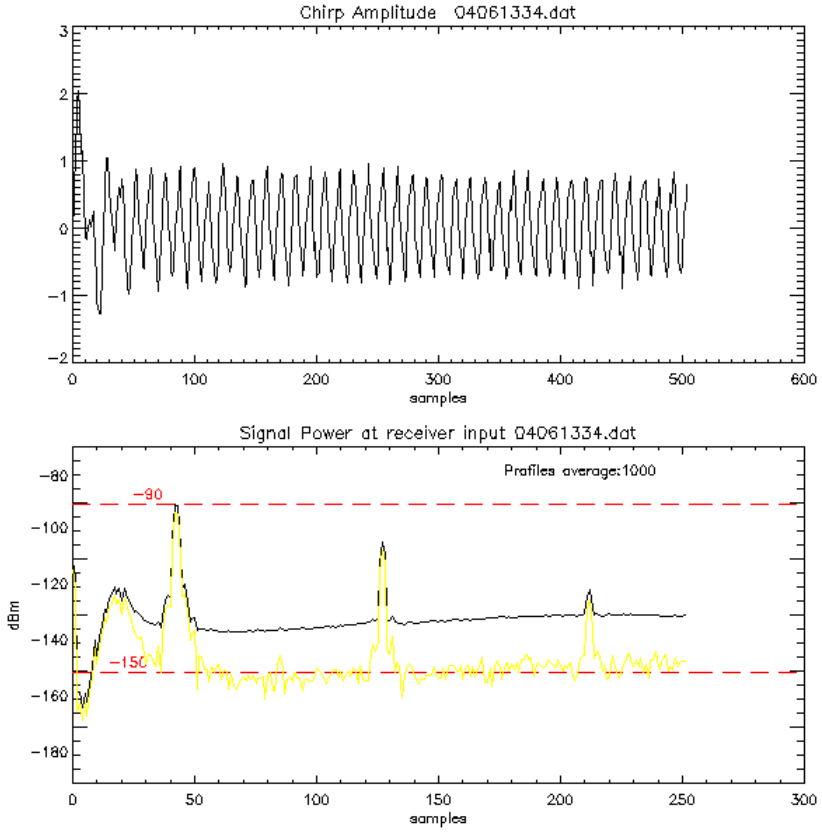


Figure 6.10: Noise floor $P_{leak} = -40.5dBm$

CHAPTER 7

FIELD DEPLOYMENTS

To verify the radar behavior in field deployment conditions, a series of test deployments were conducted during May and June 2010 in Tillson Farm. The radar hardware was placed inside the white shed in Tillson Farm, which is quipped with a generator that provides power for the radar equipment. The antennas and their shrouds were mounted on the existing flatbed trailer as shown in Figure 7.1. Two 50ft N-type cables whose losses are 2dB at 915MHz are used for connecting the antennas.

Figure 7.1: Antennas in Tillson farm

7.1 Antenna Isolation

The received signal is a mix of atmospheric echoes, clutter, and antenna leakage. In an FMCW radar, transmitter energy leaking into the receiver will produce a line in the receiver output spectrum with a frequency related to the difference between the leakage path and the local oscillator path to the mixer. For the system described in this work, the audio module filter takes care of filtering the frequency of the leakage.

However, it must be ensured that the leakage signal does not overload any part of the receiver. It is also necessary to ensure that transmitter leakage does not exceed the recommended maximum input level of the mixer, thus generating distortion products that can contaminate the entire receiver spectrum. In this case the maximum leakage is primarily determined by the audimodule. It must ensured that the amplifiers of the audio module filter are just below saturation. With the default configuration (PRF=100Hz, T_P=10ms and B=25Hz) the maximum acceptable leakage is around -50dBm. The transmit power is 44.8dBm, hence an isolation of 95dB would be needed. If attenuators are added to the input of the audiomodule, the maximum leakage can be increased but consequently the sensitivity of the wind profiler will decrease.

Figure 7.2: Received leakage

With the new shrouds, which are based on a NOAA design [14], it is possible to improve the isolation about 15dB [3]. The antenna isolation was measured using the shrouds. There was 1m of separation between the edges of the antennas. The result of the first test was a leakage of -35.6dBm in the front end (Figure 7.2), which translates into a isolation of about 80dB. Therefore, 15dB of attenuation had to be added before the audio module.

7.2 Deployment results

The atmospheric refractive index fluctuations represent a very weak target, being orders of magnitude smaller than the ground clutter. Returns from trees, buildings and other objects on the ground give rise to ground clutter and are dependent on the site. This ground clutter enters the receiver through the antenna side lobes and

decreases with the third power of range. Preferably, the Wind Profiler should be deployed in an environment as clutter free as possible.

As mentioned before, 15dB of attenuation were added, resulting in a theoretical noise floor of about -140dBm and a radar sensitivity which is too low to measure the typical values of turbulences (see Figure 7.3) .

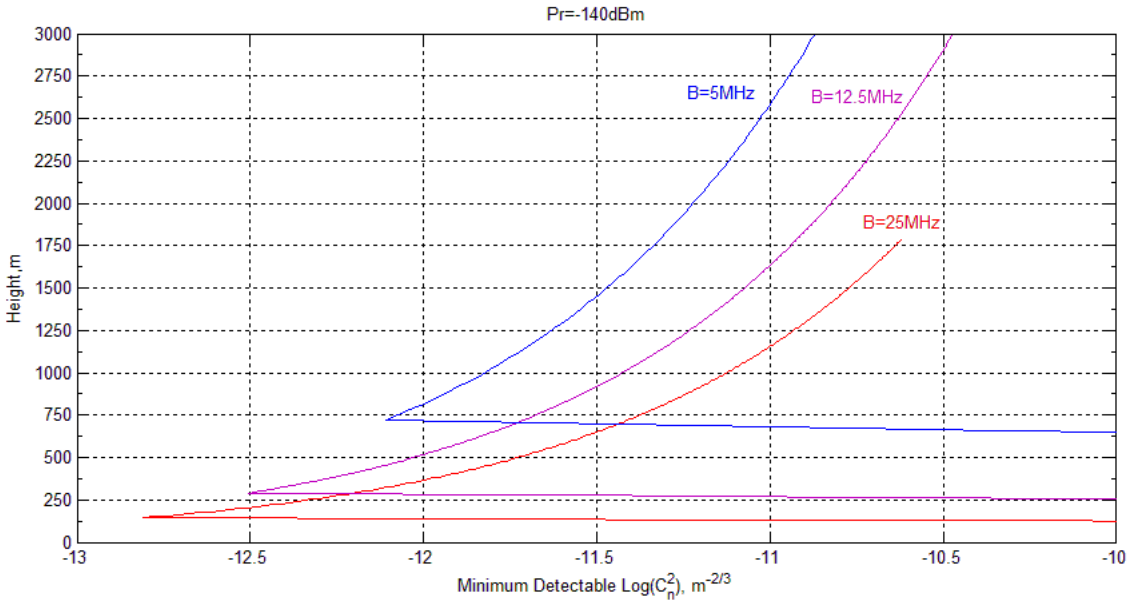


Figure 7.3: Minimum detectable C_n^2 when received power is -140dBm

Only turbulences with values of C_n^2 higher than $10^{-12}m^{\frac{-2}{3}}$ can be detected. In order to test the radar sensitivity, rain can be used as a target with higher reflectivity. The received power from a volume filled with rain drop scatterers and antennas with Gaussian-shaped beams is given by Equation 7.1. This expression illustrates that the received power is solely dependent upon radar system parameters, is proportional to the radar reflectivity factor Z, and is inversely proportional to R^2 [15].

$$P_r = \frac{P_t G^2 \theta^2 \triangle R \, \pi^3 \left|K\right|^2 Z}{512(4 \ln 2) \, R^2 \lambda^2} \tag{7.1}$$

where $\left|K\right|^2 \simeq 0.93$ at centimeter wavelengths. The calculation of Z from Equation 7.1 will have dimensions of $\frac{m^6}{m^3}$. Conversion to the more commonly used units of

$\frac{mm^6}{m^3}$ requires that the result be multiplied by a factor 10^{18}. Because the Z values of interest can range over several orders of magnitude, a logarithmic scale is often used, where $dBZ = 10log\, Z\left[\frac{mm^6}{m^3}\right]$.

Using Equation 6.1 and Equation 7.1, the relationship between Z and C_n^2 is derived as

$$C_n^2 = \frac{\pi^5 |K|^2 Z\left[\frac{mm^6}{m^3}\right] 10^{18}}{4{\cdot}0.38{\cdot}\lambda^{\frac{11}{3}}} \tag{7.2}$$

In the optically clear boundary layer, Z values of the order -20dBZ to 10dBZ ($C_n^2 \in [10^{-16}, 10^{-13}]$) are of interest. In rainy conditions, Z may range from about 20dBZ to as much as 60dBZ ($C_n^2 \in [10^{-12}, 10^{-8}]$).

During the months of April and May series of field deployments were carried out using different configurations under different weather conditions.

In the deployment of April 14th, 30 minutes of data were acquired. The sky was clear, the temperature was around 60°F with a humidity of 28%. The following images show the obtained results performing noncoherent averaging of 1sec and adding 2 consecutive chirps. Figure 7.4 using default configuration (T_P=10ms and B=25Hz) and Figure 7.5 using half bandwidth (T_P=10ms and B=12.5Hz).

The radar image shown in Figure 7.6, corresponds to 14min collected data on May 5th. For the processing, averaging of 10sec and adding 2 consecutive chirps were used to improve the sensitivity. The sky was clear with a humidity of about 40% and 70°F of temperature.

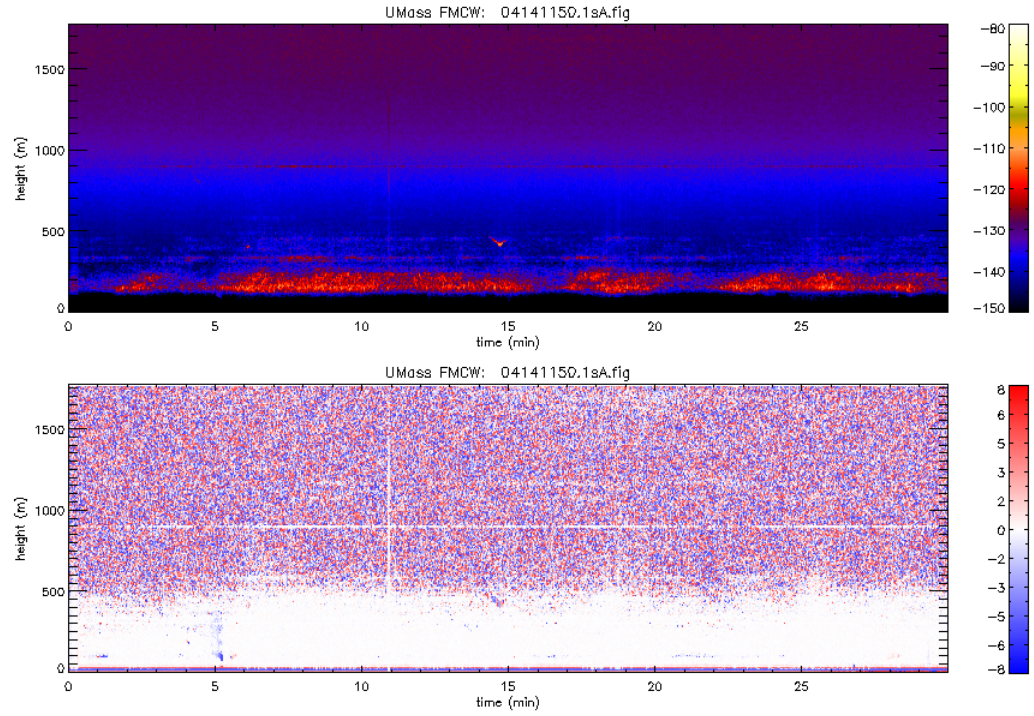


Figure 7.4: Returned power and Doppler velocity during deployment on April 14th (Default configuration)

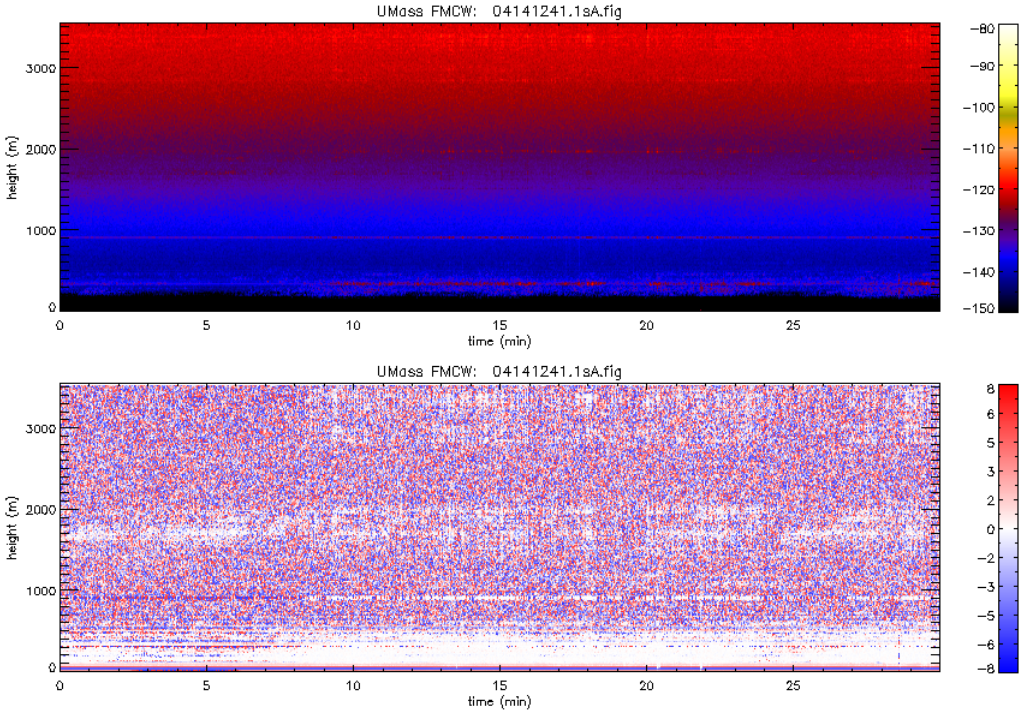


Figure 7.5: Returned power and Doppler velocity during deployment on April 14th (B=12.5MHz)

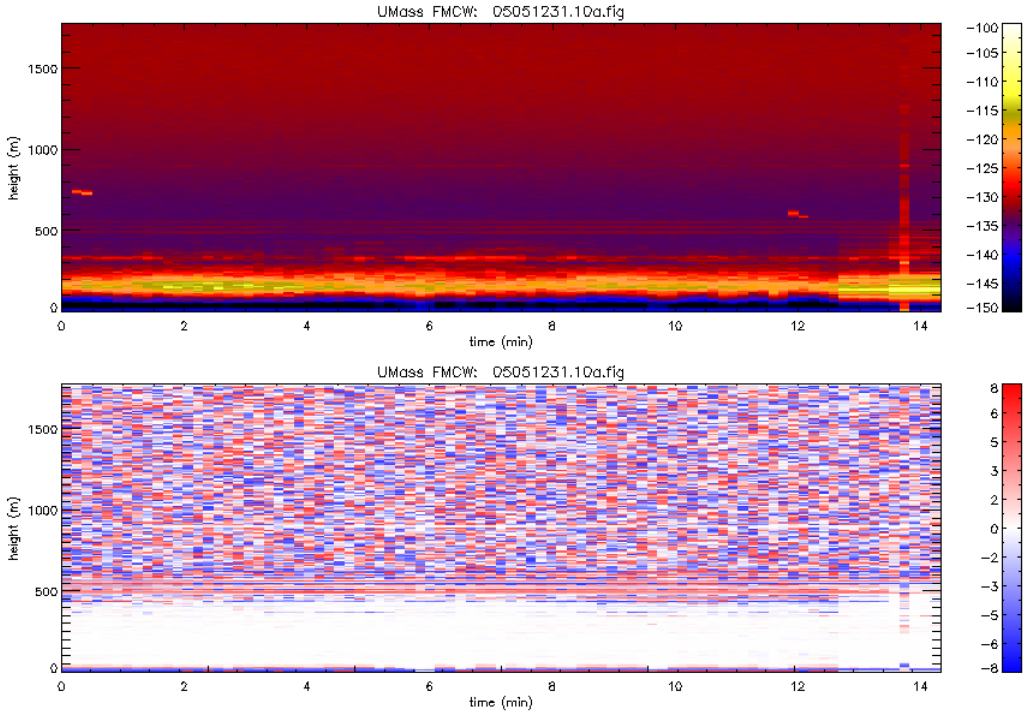


Figure 7.6: Returned power and Doppler velocity during deployment on May 5th

No discernible boundary layer features were observed in the reflectivity image. The clutter is observed in the previous images below 500m. The power spectrum of the averaging of 10,000 profiles is plotted in Figure 7.7. Knowing that the strength of radar returns from ground clutter varies with the third power of range, it can be concluded that it corresponds to ground clutter.

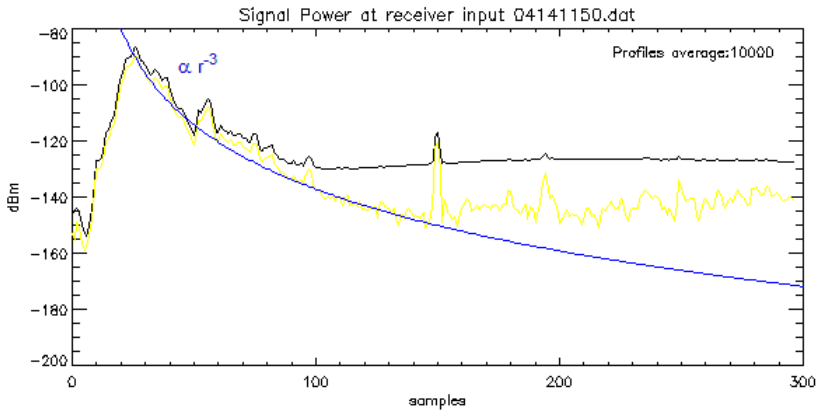


Figure 7.7: Data Power Spectrum (April 14th)

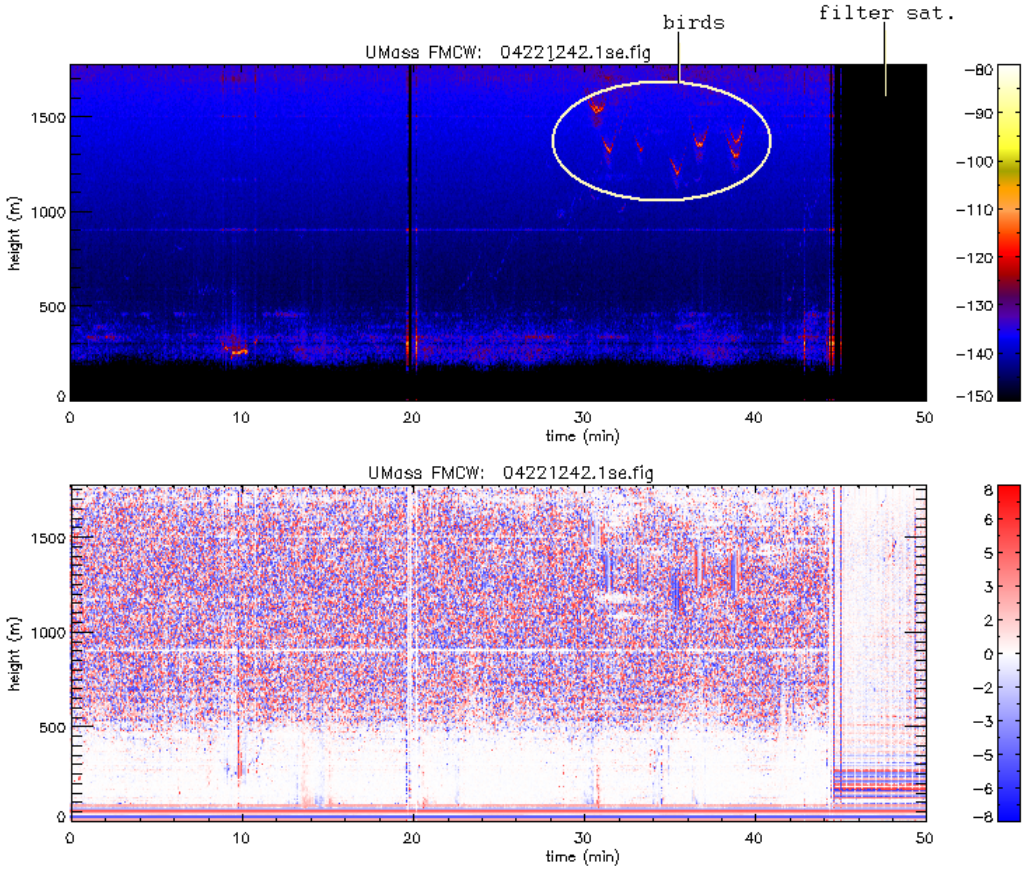


Figure 7.8: Returned power and Doppler velocity on April 22nd

The main problem of the field deployments was the instability of the audio module filter. This is an active filter and becomes saturated due to its dependence on the leakage. In the deployment of April 22nd, 50min data was acquired using T_P=10ms and $f_{Sample} = 30kHz$. At the time of the data collection the sky was partly cloudy and the temperature around 65°F. Unfortunately, when it started drizzling, the filter saturated

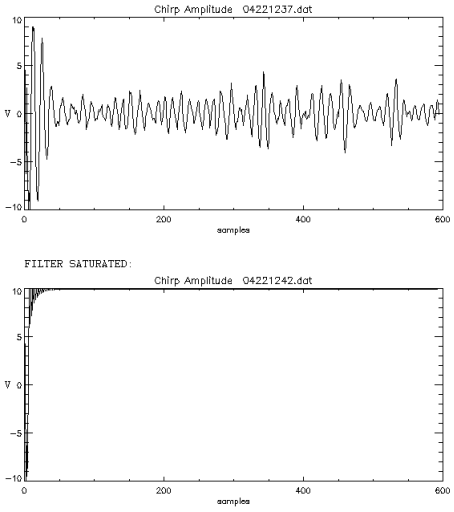


Figure 7.9: Audio module output

(see Figure 7.8). The chirp at the output of the audio module is shown on Figure 7.9. The same situation occurred using different configurations in several deployments when it was trying to acquire data while it was raining. More attenuation was added not to saturate the filter, resulting in a noise floor which was not good enough to detect the atmospheric turbulences.

Figure 7.10: Antennas with additional shrouds

To improve the isolation between transmitting and receiving antennas for the wind profiler, additional shrouds were added to the current antennas as shown in Figure 7.10. Using these shrouds, the antenna isolation improved by 5dB. Thus only 10dB of attenuation were necessary at the audio module input. With this configuration data was acquired on May 12th. The sky was cloudy with a humidity of 80% and temperature of 44°F. The result is shown on Figure 7.11, using the default configuration, adding 2 consecutive chirps and averaging of 10sec. A cloud can be observed in the radar image between 600-800m of height.

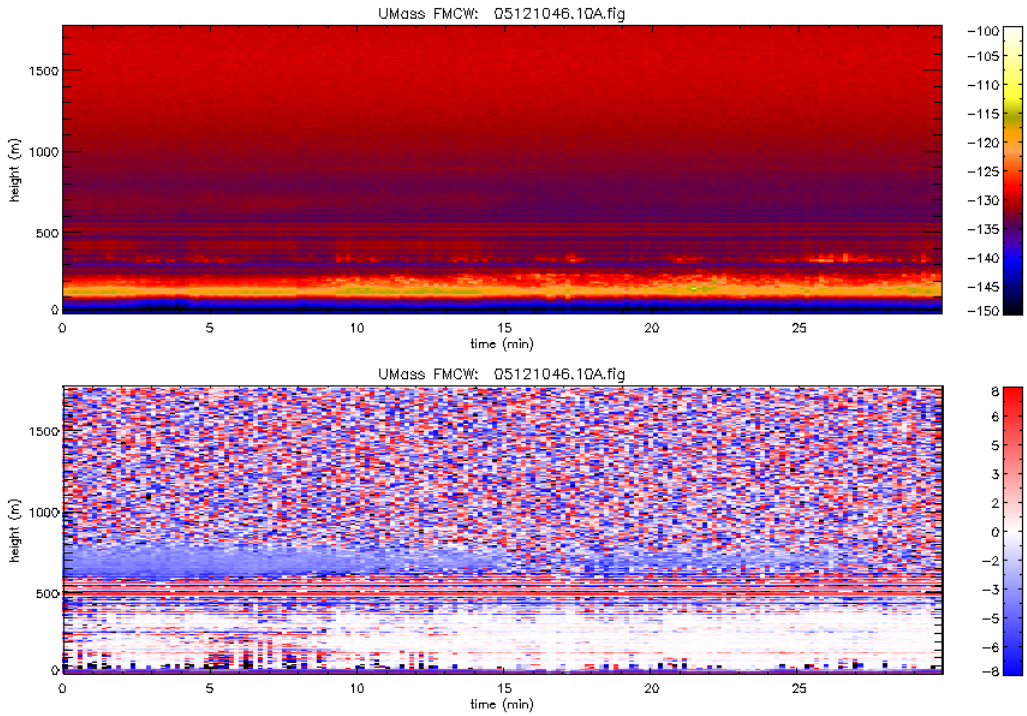


Figure 7.11: Returned power and Doppler velocity during deployment on May 12th

CHAPTER 8

FUTURE WORK

FMCW radars offer many advantages such as low peak power, low probability of interception, low interference with other systems, and high-range resolution. However, their major drawback is the isolation required between the transmitter and receiver. A scheme has been proposed to enable FMCW radar to operate with a single antenna using frequency-modulated interrupted continuous-wave (FMICW) technology. The wind profiler at later stage will utilize spaced antenna (SA) technique for achieving three-dimensional wind field.

8.1 FMICW configuration

As commented before, high transmit/receive isolation requirements are difficult to achieve. For this reason a FMICW design has been proposed. In this section it is described the configuration of the UHF FMCW radar that has been adapted for single antenna operation by alternately switching the single antenna between the transmitter and the receiver. A simplified scheme of the system is given in Figure 8.1.

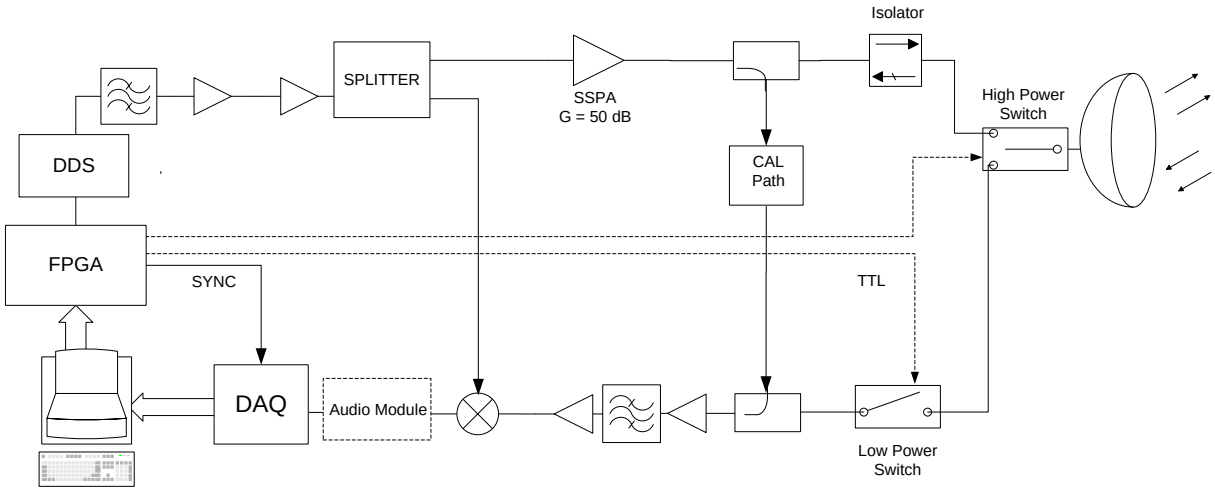


Figure 8.1: FMICW Block Diagram

This scheme works with two PIN diodes switches, a high power switch and a low power switch. Considering isolation constraints and the performance of the wind profiler, both have been chosen from available commercial PIN diode switches. The high power switch from Micronetics, Inc. is a Single Pole Double Throw (SPDT) for T/R. It is reflective, which means that power presented at any OFF port will be reflected. For this reason an isolator is located in the system. It is designed for reliable operation at power levels of 50W averages, thus it is good enough for the requirement of the transmitted power (30W). It offers high isolation, fast switching speed and low loss. Table 8.1 lists the basic specifications.

Frequency Range	0.1 to 1 GHz
Power, CW (Maximum)	50 Watts
Isolation (Minimum)	50 dB
Insertion Loss (Maximum)	0.75 dB
RF Switching Speed (Maximum)	200 ns
Control Switching Freq. (Maximum)	100 KHz
DC Power Requirements	
Positive Supply	+5 +/-2% Vdc
Positive Supply Current (Max)	150 mA
Negative Supply Voltage	-70 +/-2% Vdc
Negative Supply Current (Max)	150 mA

Table 8.1: High Power Switch SPDT (Ref. ES0309-50)

The Low Power SPST Diode switch is from Mini-Circuits. It is absorptive, and it provides high isolation and very high switching speed. Its specifications are given in Table 8.2.

In both cases the necessary switching voltages are derived from TTL signals (without built-in drivers), these signals have to be generated by the FPGA. In order to allow for the switching time and to achieve the required isolation, a dead time of the order of nanoseconds must be provided between transmit and receive phases of the switching. This ensures that the transmitter is turned fully off before the receiver starts to turn on. The required power supplies for these switches will have to be

Frequency Range	DC to 2000 MHz
RF Power Input Max.	+33 dBm
Isolation (Minimum)	58 dB
Insertion Loss (Typical)	1.3 dB
Rise/Fall Time (10%-90%) (Max.)	5 ns
Switching Time (Typical)	
50% of control to 90% RF (Turn-on)	7 ns
50% of control to 10% RF (Turn-off)	3 ns
Control Voltage	
Low State	-0.2 to 0 V
High State (negative)	-5 to -8 V
Control Current	
0.2mA Max. to -8V	
0.5mA Max. at -9V to -12V Typ.	

Table 8.2: Low Power Switch SPST (Ref. ZFSWHA-1-20+)

included in the radar hardware.

The switching induces a constraint on R_{MAX}, given by Equation 2.15. As commented in Section 2.3 the rectangular wave which finally modulates an echo depends on the target range. The limit on the maximum range can be increased by reducing the duty cycle of the T/R waveform, at the expense of reduced signal to noise ratio. However, this scheme has the additional advantage of introducing a desirable range sensitivity characteristic. The system has its best performance when the choice of the switching frequency is given by Equation 2.12. By this way, the power of the closest ranges will be significantly reduced, but the farthest targets, more attenuated due to propagation losses, have the highest sensitivity. In this case the achieved isolation is good enough for the wind profiler. On Table 6.1 FMICW parameters are calculated for different configurations.

Unlike FMCW it is not possible to choose any value for the sampling rate as it is determined by Equation 2.14. An antialiasing filter is still necessary to prevent aliasing when the received signal is sampled by the ADC but the filter must also remove the higher order copies of the echo spectrum produced by the switching. That

T_P	**10 ms**	**10 ms**	**10 ms**	**5 ms**	**5 ms**	**20 ms**
B	**25 MHz**	**12.5 MHz**	**5 MHz**	**25 MHz**	**12.5 MHz**	**25 MHz**
ΔR	6 m	12 m	30 m	6 m	12 m	6 m
$R_{max,FMICW}$	1500 m	2121 m	3354 m	1061 m	1500 m	2121 m
$f_S(>f_{Sample})$	50 kHz	35.36 kHz	22.36 kHz	70.71 kHz	50 kHz	35.36 kHz
$v_{r,\,max}$	8.2 m/s	8.2 m/s	8.2 m/s	16.4 m/s	16.4 m/s	4.1 m/s
$R_{CAL} = 300m$						
$f_{b,CAL}$	5 kHz	2.5 kHz	1 kHz	10 kHz	5 kHz	2.5 kHz
$f_{c,filter} \simeq 2.5kHz$						
R_{min}	150 m	300 m	750 m	75 m	150 m	300 m
$R_{leak,lab} = 12.6m$						
f_{Leak}	210 Hz	105 Hz	42 Hz	420 Hz	210 Hz	105 Hz

Table 8.3: FMICW parameters

is not a big issue since the ADC provides the antialiasign filter, only it is necessary to set the appropriate sample frequency f_{Sample}.

8.2 Spaced Antenna

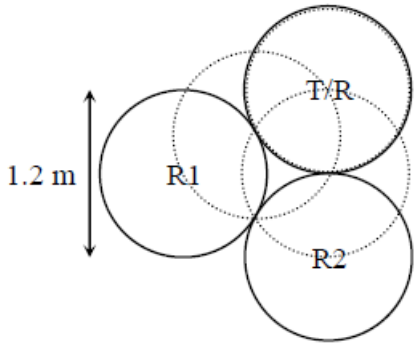


Figure 8.2: 3-antenna SA-FMICW

As commented before the radar now operates with one receiving antenna, but the final design will utilize spaced-antenna technique to measure horizontal winds, hence providing information about 3D wind fields. This technique can obtain more rapid wind estimates compared to Doppler beam swinging systems [16]. It is planned to use three antennas: one transmit/receive antenna and two receive antennas arranged in a triangle, as shown in Figure 8.2. Solid circles denote physical apertures, while

dashed lines denote virtual apertures. They would form an effective antenna whose phase center is between the physical antennas.

CHAPTER 9

CONCLUSIONS

The work presented in this thesis is a description of the the FMCW Wind Profiler as of Spring 2010. A new Data Acquisition Card was integrated to increase the dynamic range of the radar. New data collection programs were developed for it, solving the synchronization issues. Signal processing programs have also been developed to retrieve and display the reflectivity power and the Doppler velocity of the wind profiler. Since all the required operations to manage the radar, e.g. initialization of the acquisition card and data processing, are quite complex, a graphical user interface has been created to make the operator tasks easier. Moreover, to avoid waits to acquire all the data to start the processing, real time programs were developed too, which allow displaying the reflectivity power while data is being acquired.

A series of tests were carried out in the laboratory. The maximum sensitivity the radar can reach was determined. For this purpose the noise floor was measured using the calibration signal as a reference. A needed extra amplifier was added in the front end. This way, the noise floor is primarily determined by the front end instead of the audio module. As a consequence of adding this amplifier, the maximum leakage acceptable was decreased, which translates into a high value of the necessary isolation.

Following this, the isolation was measured in the first field deployment using the shrouds. To do the field measurements, attenuators had to be added before the audio module, because this is the first component to become saturated. Thereby, the sensitivity of the radar was reduced. The results of the field deployments have been discussed. The audio module could be modified in the future since it was designed for the old DAQ, and currently the antialiasing filter is already provided by the ADC. To improve the isolation it could use larger antennas and therefore, greater gain. But this would make the use of spaced antenna technique difficult.

The isolation requirements are difficult to achieve. To solve that, the use of frequency-modulated interrupted continuous-wave is planned, as well as reducing the number of antennas from four to three. The new scheme has been described and the new parameters of the wind profiler have been calculated. Finally, the use of a spaced antenna technique is also planned to add the ability to analyze horizontal winds.

BIBLIOGRAPHY

[1] Muschinski A., Turker I., Frasier S. and Pazmany A. An S-band FMCW boundary layer profiler: Description and initial results. *Radio Science 38*, 2003.

[2] David Garrido Lopez. An UHF Frequency-Modulated Continous Wave Wind-Profiler - Receiver and Audio Module Development. *Master's Thesis at UMASS Amherst*, 2009.

[3] David Garrido Lopez. An UHF Frequency-Modulated Continous Wave Wind-Profiler - Developement and Initial Results. *Master's Thesis at UMASS Amherst*, 2009.

[4] David Atlas. Radar in Meteorology. *American Meteorological Society, Boston*, 1990.

[5] Ottersen H. Atmospheric structure and radar backscattering in clear air. *Radio Science 4*, **12**:1179–1193, 1969.

[6] J.A. McGregor, E.M. Poulter, M.J. Smith. Switching system for single antenna operation of an S-band FMCW Radar. *IEE Proceedings- Radar, Sonar and Navigation*, **141**(4):241–248, 1994.

[7] Albert Genis. Implementation and test of an FMCW Wind Profiler Transceiver. *Master's Thesis at UMASS Amherst*, 2008.

[8] Iva Kostadinova. An UHF Frequency-Modulated Continous Wave WindProfiler - Developement and Initial Results. *Master's Thesis at UMASS Amherst*, 2009.

[9] General Standards Corporation. PC104P-24DSI6LN Specifications.

[10] General Standards Corporation. PC104P-24DSI6LN Reference Manual.

[11] General Standards Corporation. 24DSI Linux Device Driver User Manual. 2009.

[12] ITT Visual Information Solutions. Programming in IDL.

[13] Richard J. Doviak and Dusan S. Zrnic. Doppler Radar and Weather Observations. *Dover Publications, Inc. Mineola New York*, 1984.

[14] Paul E. Johnston, David A. Carter, James R. Jordan, Allen B. White. A New Snow-Level Detection Radar. *NOAA*, 2009.

[15] Merrill Skolnik. Radar handbook. *McGraw Hill, Boston*, 1990.

[16] Muschinski A., S.J. Frasier. Supplement to 'Investigation of Turbulence and Intermittency in the Convective Boundary Layer Using a 915MHz Volume Wind Profiler': Profiler Fabrication. 2006.

A.1 Savedata_Windprof

The source code used to collect data can be found in `savedata_windprof` folder and it is detailed below.

A.1.1 main.c

> Currently the program only acquires data from channel 0, to acquire also channel 1 and 2, it is required to comment the following line:
> "251 if((l%4==0)|(l==0)) {//Now only we want chan 0"
> Don't forget in windprof_process.pro change the variable which indicates the number of channels in raw file (number_channels=1 -> 4).

```
1  #include <ctype.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <string.h>
5  #include <time.h>
6  #include "main.h"
7  #include "fmcw_conf.c"
8  #define BLOCK_SIZE_MAX  2000L
9  #define CHANS_ENABLED    4
10 #define THRESHOLD 60063000
11
12 // variables ***************************************************
13 static   int      _continuous    = 0;
14 static   int      _ignore_errors = 0;
15 static   int      _index         = 0;
16 static   long  _minute_limit  = 0;
17 static   __s32 _buffer1[1L];
18 static   __s32 _buffer_data[BLOCK_SIZE_MAX];
19 /*************************************************************
20 *  Function:   _parse_args
21 *  Purpose: Parse the command line arguments.
22 *  Arguments:
23 *     argc  The number of command line arguments given.
24 *     argv  The list of command line arguments given.
25 *  Returned:   >= 0  The number of errors encounterred.
26 *************************************************************/
27
28 static int _parse_args(int argc, char** argv)
29 {
30    char  c;
31    int      errs  = 0;
32    int      i;
33    int      j;
34    int      k;
35    printf("USAGE: savedata <-c> <-C> <-m#> <-n#> <index>\n");
36    printf("   -c     Continue testing until an error occurs.\n");
37    printf("   -C     Continue testing even if errors occur.\n");
```

```
38     printf("   -m#    Run for at most # minutes (a decimal number).\n");
39     printf("   index  The index of the board to access.\n");
40     printf(" NOTE: Hit Ctrl-C to abort continuous testing.\n");
41
42     for (i = 1; i < argc; i++)
43     {
44        if (strcmp(argv[i], "-c") == 0)
45        {
46           _continuous    = 1;
47           _ignore_errors = 0;
48           continue;
49        }
50        if (strcmp(argv[i], "-C") == 0)
51        {
52           _continuous    = 1;
53           _ignore_errors = 1;
54           continue;
55        }
56        if ((argv[i][0] == '-') && (argv[i][1] == 'm') && (argv[i][2]))
57        {
58           j  = sscanf(&argv[i][2], "%d%c", &k, &c);
59
60           if ((j == 1) && (k > 0))
61           {
62              _minute_limit  = k;
63              continue;
64           }
65           errs  = 1;
66           printf("ERROR: invalid argument: %s\n", argv[i]);
67           break;
68        }
69        j= sscanf(argv[i], "%d%c", &k, &c);
70        if ((j == 1) && (k >= 0))
71        {
72           _index   = k;
73           continue;
74        }
75        else
76        {
77           errs  = 1;
78           printf("ERROR: invalid board selection: %s\n", argv[i]);
79           break;
80        }
81     }
82     return(errs);
83  }
84
85  /*************************************************************
86  *  Function:   _perform_tests
87  *  Purpose: Perform the appropriate testing.
88  *  Arguments:  fd The handle for the board to access.
89  *  Returned:   >= 0  The number of errors encounterred.
90  *************************************************************/
91
92  static int _perform_tests(int fd)
93  {
94     int         errs  = 0;
95     const char* psz;
96     struct tm*  stm;
97     time_t      tt;
98     time(&tt);
99     stm   = localtime(&tt);
100    psz   = asctime(stm);
101    gsc_label("Performing Operation");
102    printf("%s", psz);
103    errs  += gsc_id_driver(fd, DSI_BASE_NAME); //Identify the driver
104    errs  += dsi_id_board(fd, -1, NULL); //Identify the board
105    errs  += save_data(fd);
```

```
106     return(errs);
107  }
108
109  /*************************************************************
110  *     Function:   findedge
111  *  Purpose: find positive or negative edge
112  *  Arguments:  aux   (previous state sample '0' or '1')
113  *        current_sample
114  *  Returned:   1 positive edge, 2 negative edge
115  *        0 no edge
116  *************************************************************/
117  int findedge(int aux, long current_sample)
118  {
119     if (aux==0)//Currently low level,start looking for a positive edge
120     {
121        if (current_sample>THRESHOLD)
122        {
123        return(1);//Positive edge found
124        }
125     }
126     else //(aux==1) continue looking for low level
127     {
128        if (current_sample<THRESHOLD)
129        {  return(2);//Negative edge found
130        }
131     }
132  return(0);
133  }
134
135  /*************************************************************
136  *  Function:   main
137  *  Purpose: Control the overall flow of the application.
138  *  Arguments:  argc  The number of command line arguments.
139  *        argv  The list of command line arguments.
140  *  Returned:   EXIT_SUCCESS   We tested a device.
141  *        EXIT_FAILURE   We didn't test a device.
142  *************************************************************/
143
144  int main(int argc, char *argv[])
145  {
146     int   errs;
147     int   fd= 0;
148     FILE* fd_file;
149     time_t   exec= time(NULL);
150     long  hours,mins,secs;
151     time_t   starttime,now,t_limit,test;
152     struct tm *today;
153     __s32 chans = 32;
154     int   qty;
155     int   ret= EXIT_FAILURE;
156     char datebuf[128];
157     char filename[128];
158     int got;
159     int start=0;
160     int cont_profiles=0;
161     int aux;
162     int flag_edge=0;
163     int cont=0;
164     int cont_writing=0;
165     int l,i;
166     gsc_label_init(26);
167     int number_total_profiles;
168     test  = time(NULL);
169     printf("savedata - Capture and Save Data to Disk by Carol (Version %s)\n", VERSION
          );
170     errs  = _parse_args(argc, argv);
171          long sample_rate;
172          long Block_Size;
```

```
173          long TamBuffer;
174
175         //reading header
176     struct fmcw_conf_struct fmcw_conf;
177     printf("fmcw: reading configuration file\n");// Read FMCW configuration file
178     if (read_fmcw_conf("/home/profiler/Desktop/windprofiler2010/fmcw_current.conf", &
            fmcw_conf)) {
179       fprintf(stderr, "fmcw: configuration file error\n");
180     }
181     number_total_profiles=fmcw_conf.number_of_profiles;
182     sample_rate=fmcw_conf.sample_frequency*1000L;
183          Block_Size=(fmcw_conf.chirp_length*fmcw_conf.sample_frequency-5);
184          TamBuffer= Block_Size*4;
185          //printf("TamBuffer %ld and sizeof(_buffer_data)=%ld\n",TamBuffer ,sizeof(
                 _buffer_data));
186     gsc_id_host();//Identify the host os and machine
187     t_limit  = exec + (_minute_limit * 60);
188     qty   = gsc_count_boards(DSI_BASE_NAME);// Count the number of installed SI04
189     errs  = gsc_select_1_board(qty, &_index); //Select the board to utilize when more
            than one is present
190        if ((qty <= 0))      errs++;
191        gsc_label("Accessing Board Index");
192        printf("%d\n", _index);
193        fd= gsc_dev_open(_index, DSI_BASE_NAME); //Perform an open on the device with
               the specified index.
194        if (fd == -1)//finish testing and start acquiring data
195        if(!errs || (_ignore_errors == 0))
196        {
197           fd = gsc_dev_open(_index, DSI_BASE_NAME);
198           dsi_config_board(fd, -1, -1, sample_rate);
199           errs += _channels_profiler(fd, &chans);
200           //Creating the file
201           printf("Creating the file where the data will be stored\n");
202           time(&starttime);today = localtime(&starttime);
203              strftime(datebuf, 128, "%m%d%H%M", today);
204              sprintf(filename, "/home/profiler/Desktop/windprofiler2010/data/%s.dat",
                     datebuf);
205           fd_file =fopen(filename, "w+b");
206           if (fd_file == -1)    errs= 1;
207           else printf("file created: %s\n",filename);
208        fwrite_fmcw_conf(fmcw_conf, fd_file);//write header to file
209        printf("Starting acquisition!!!!!!!!!!!!!!!!\n Wait...\n");
210        for(;;)
211        {
212           now= time(NULL);
213           errs = 0;
214        if(start==0)//to figure out which is the 1st state (aux= 0 or 1?)
215        {
216              got = dsi_dsl_read(fd, _buffer1, sizeof(_buffer1));//Read first sample
217              if (got < 0)   errs= 1;
218              else     errs= 0;
219              if(_buffer1[0]==8388608) start=0; //no count 1st samples 0 ...
220              else {
221              start=1;
222              /* Notes:
223                _buffer1[0] corresponds to chan 0 -- DATA
224                _buffer1[3] corresponds to chan 3 -- SYNC
225              */
226              if (_buffer1[3]<THRESHOLD) aux=0;
227              else if (_buffer1[3]>THRESHOLD)        aux=1;
228              }
229        }else{//if(start==0)
230           got = dsi_dsl_read(fd, _buffer1, sizeof(_buffer1));//Read next sample
231           if (got < 0)   errs= 1;
232           else     errs= 0;
233           flag_edge=findedge(aux, _buffer1[3]);
234           if(flag_edge==1) {
235                   aux=1;
```

```
                cont_writing=1;
                cont=0;}
         if(flag_edge==2) {aux=0;
                }
      if(cont_writing==1)  //Save Block_Size samples
      {
          //Read next Blocksize & Writing block
         for(l=0; l< TamBuffer; l++) _buffer_data[l]=0;
         got = dsi_dsl_read(fd, _buffer_data, TamBuffer);
         if (got < 0) errs= 1;
         else if (got != TamBuffer)
         {  printf("There aren't a full block \n");
         }
         else{
            for(l=0; l< TamBuffer; l++)
            {  if((l%4==0)|(l==0)) {//Now only we want chan 0
               //printf("got= %d, sample read buffer1=%08lX \n",got,_buffer_data[l]);
               i = fwrite( &_buffer_data[l],1, 4,fd_file);
               cont++;
               }
            }
            //printf("Block read:%d samples\n",cont);
            cont_writing=0;
            cont_profiles++;
         }
      }//if(cont_writing==1)
      }//if(start==0)
      if ((_minute_limit) && (now >= t_limit)) break;
      if ((_ignore_errors == 0) && (errs))    break;
      if(cont_profiles==number_total_profiles) break;
      }//for(;;)

      if(cont_profiles < number_total_profiles) number_total_profiles=cont_profiles;
      printf("Total number of profiles:%d \n ",number_total_profiles);

      if(errs==0)  printf("the end OK :) \n ");
           else  printf("the end with some error :( \n");

      gsc_dev_close(_index, fd);
      fclose(fd_file);
      }
   return(ret);
}
```

A.1.2 savedata.c

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "main.h"
// #defines ***************************************************************
#define  _1M      (1024L * 1024L)
#define TRIGGER_RATE     100
#define SAMPLE_RATE     60000L
#define CHANS_ENABLED    4
// variables   ************************************************************
static   __u32 _buffer[(SAMPLE_RATE * CHANS_ENABLED) / TRIGGER_RATE];
/***************************************************************************
*
*  Function:   _channels
```

```
17  *  Purpose:
18  *     Check to see how many channels the board has.
19  *  Arguments:
20  *     fd    The handle for the board to access.
21  *     chan  Report the number of channels here.
22  *  Returned:
23  *     >= 0  The number of errors encounterred.
24  ****************************************************************************/
25  static int _channels(int fd, __s32* chans)
26  {
27     int   errs;
28     gsc_label("Input Channels");
29     errs  = dsi_query(fd, DSI_QUERY_CHANNEL_QTY, chans);
30     if (errs == 0)
31        printf("%ld Channels\n", (long) chans[0]);
32     return(errs);
33  }
34
35  /****************************************************************************
36  *  Function:   _read_data
37  *  Purpose:
38  *     Read data into the buffer.
39  *  Arguments:
40  *     fd    The handle for the board to access.
41  *  Returned:
42  *     >= 0  The number of errors encounterred.
43  ****************************************************************************/
44
45  static int _read_data(int fd)
46  {
47     int      errs;
48     long  get      = sizeof(_buffer) / CHANS_ENABLED;
49     int      got;
50     gsc_label("Reading");
51     got   = dsi_dsl_read(fd, _buffer, get);
52
53     if (got < 0)
54     {
55        errs  = 1;
56     }
57     else if (got != get)
58     {
59        errs  = 1;
60        printf(  "FAIL <---  (got %ld samples, requested %ld)\n",
61              (long) got,
62              (long) get);
63     }
64     else
65     {
66        errs  = 0;
67        printf(  "PASS  (%ld samples)\n",
68              (long) get);
69     }
70
71     return(errs);
72  }
73
74  /****************************************************************************
75  *  Function:   _save_data
76  *  Purpose:
77  *     Save the read data to a text file.
78  *  Arguments:
79  *     fd    The handle for the board to access.
80  *     chan  The number of channels.
81  *     errs  have there been any errors so far?
82  *  Returned:
83  *     >= 0  The number of errors encounterred.
84  ****************************************************************************/
```

```
85
86  static int _save_data(int fd, int chan, int errs)
87  {
88     FILE*    file;
89     int         i;
90     long     l;
91     const char* name  = "data.txt";
92     long     samples  = sizeof(_buffer) / CHANS_ENABLED;
93     gsc_label("Saving");
94
95     for (;;)
96     {
97        if (errs)
98        {
99           printf("SKIPPED  (errors)\n");
100          errs  = 0;
101          break;
102       }
103       file  = fopen(name, "w+b");
104       if (file == NULL)
105       {
106          printf("FAIL <---  (unable to create %s)\n", name);
107          errs  = 1;
108          break;
109       }
110
111       for (l = 0; l < samples; l++)
112       {
113          i  = fprintf(file, "  %08lX", (long) _buffer[l]);
114          if (i != 10)
115          {
116             printf("FAIL <---  (fprintf() failure to %s)\n", name);
117             errs  = 1;
118             break;
119          }
120          if ((l % chan) == (chan - 1))
121          {i = fprintf(file, "\r\n");
122             if (i != 2)
123             {
124                printf("FAIL <---  (fprintf() failure to %s)\n", name);;
125                errs  = 1;
126                break;
127             }
128          }
129       }
130       fclose(file);
131       if (errs == 0)
132          printf("PASS  (%s)\n", name);
133       break;
134    }
135    return(errs);
136 }
137
138
139 /*****************************************************************************
140 *  Function:   save_data
141 *  Purpose:
142 *     Configure the board, then capture data to a file.
143 *  Arguments:
144 *     fd    The handle for the board to access.
145 *  Returned:
146 *     >= 0  The number of errors encounterred.
147 *****************************************************************************/
148
149 int save_data(int fd)
150 {
151    __s32 chans = 32;
152    int      errs  = 0;
```

```
153
154     gsc_label("Capture & Save");
155     printf("\n");
156     gsc_label_level_inc();
157
158     errs  += dsi_config_board_profiler(fd, -1, -1, SAMPLE_RATE);
159     errs  += _channels(fd, &chans);
160     errs  += _read_data(fd);
161     errs  += dsi_ain_buf_overflow(fd, -1, DSI_AIN_BUF_OVERFLOW_TEST, NULL);
162     errs  += dsi_ain_buf_underflow(fd, -1, DSI_AIN_BUF_UNDERFLOW_TEST, NULL);
163     errs  += _save_data(fd, chans, errs);
164
165     gsc_label_level_dec();
166     return(errs);
167 }
168
169 /*************************************************************************
170 *  Function:   dsi_config_board_profiler
171 *  Purpose:
172 *  Configure the given board using common FMCW defaults.
173 *  Arguments:
174 *     fd    The handle to use to access the driver.
175 *     index The index of the board to access. Ignore if < 0.
176 *     fref  This is the PLL Fref value, or -1 to use the default.
177 *     fsamp This is the desired Fsamp rate, of -1 to use the default.
178 *  Returned:
179 *     >= 0  The number of errors encountered here.
180 *************************************************************************/
181
182 int   dsi_config_board_profiler(int fd, int index, __s32 fref, __s32 fsamp)
183 {
184     int      errs  = 0;
185     __s32 ndiv;
186     __s32 nrate;   // Legacy
187     __s32 nref; // PLL
188     __s32 nvco; // PLL
189     errs  += dsi_rx_io_mode          (fd, index, GSC_IO_MODE_DMA, NULL);
190     errs  += dsi_rx_io_overflow         (fd, index, DSI_IO_OVERFLOW_CHECK, NULL);
191     errs  += dsi_rx_io_timeout       (fd, index, 30, NULL);
192     errs  += dsi_rx_io_underflow        (fd, index, DSI_IO_UNDERFLOW_CHECK, NULL);
193     errs  += dsi_initialize          (fd, index);
194     errs  += dsi_ain_mode               (fd, index, DSI_AIN_MODE_DIFF, NULL);
195     errs  += dsi_ain_range           (fd, index, DSI_AIN_RANGE_10V, NULL);
196     errs  += dsi_ain_buf_input       (fd, index, DSI_AIN_BUF_INPUT_ENABLE, NULL);
197     errs  += dsi_ain_buf_threshold      (fd, index, 128L * 1024, NULL);
198     errs  += dsi_sw_sync_mode           (fd, index, DSI_SW_SYNC_MODE_CLR_BUF, NULL);
199     errs  += dsi_channel_order       (fd, index, DSI_CHANNEL_ORDER_SYNC, NULL);
200     errs  += dsi_data_format            (fd, index, DSI_DATA_FORMAT_OFF_BIN, NULL);
201     errs  += dsi_data_width          (fd, index, DSI_DATA_WIDTH_24, NULL);
202     errs  += dsi_init_mode           (fd, index, DSI_INIT_MODE_INITIATOR, NULL);
203     errs  += dsi_external_clock_source  (fd, index, DSI_EXT_CLK_SRC_GRP_0, NULL);
204     errs  += dsi_fref_compute           (fd, index, 0, &fref);
205     errs  += dsi_channel_group_source_all  (fd, index, DSI_CH_GRP_SRC_GEN_A, 1); //
            DSI_CH_GRP_SRC_GEN_A
206     errs  += dsi_fsamp_compute       (fd, index, 0, 0, fref, &fsamp, &nvco, &nref, &
            nrate, &ndiv);
207     errs  += dsi_rate_gen_x_nvco_all    (fd, index, nvco, 1);
208     errs  += dsi_rate_gen_x_nref_all    (fd, index, nref, 1);
209     errs  += dsi_rate_gen_x_nrate_all      (fd, index, nrate, 1);
210     errs  += dsi_rate_gen_x_ndiv_all    (fd, index, ndiv, 1);
211     errs  += dsi_fsamp_report_all       (fd, index, 1, &fref);
212     errs  += dsi_xcvr_type        (fd, index, DSI_XCVR_TYPE_LVDS, NULL);
213     errs  += dsi_auto_calibrate         (fd, index);
214     errs  += dsi_ain_buf_clear_at_boundary(fd, index);
215     errs  += dsi_ain_buf_overflow       (fd, index, DSI_AIN_BUF_OVERFLOW_CLEAR, NULL);
216     errs  += dsi_ain_buf_underflow      (fd, index, DSI_AIN_BUF_UNDERFLOW_CLEAR, NULL)
            ;
217
```

```
218     return(errs);
219  }
220
221  /*************************************************************************
222  *  Function:   _save_data_profiler
223  *  Purpose:
224  *     Save the read data to a file, customized for te FMCW radar
225  *  Arguments:
226  *     Buffer      The data to store
227  *     chan     The number of channels
228  *     fd_file     Th handle of the file where to store the data
229  *  Returned:
230  *     >= 0  The number of errors encountered here.
231  *************************************************************************/
232
233  int _save_data_profiler(FILE* fd_file, int chan, __s32 *buffer, long samples)
234  {
235     int i;
236     long l;
237     //long samples = sizeof(buffer)/CHANS_ENABLED;
238     int errs = 0;
239     //gsc_label("Saving data profiler\n");
240     for(l=0; l< samples; l++)
241     {
242        i = fprintf(fd_file, "  %08lX", (long) buffer[l]);
243        //i = fwrite( &buffer[l],1, sizeof(buffer[l]),fd_file);
244                  //printf("Sample %i %ld\n", i,(long) buffer[l]);
245     }
246     return(errs);
247  }
248
249  /*************************************************************************
250  *  Function:   _channels_profiler
251  *  Purpose:
252  *     Check to see how many channels the board has.
253  *  Arguments:
254  *     fd    The handle for the board to access.
255  *     chan  Report the number of channels here.
256  *  Returned:
257  *     >= 0  The number of errors encounterred.
258  *************************************************************************/
259
260  int _channels_profiler(int fd, __s32* chans)
261  {
262     int   errs;
263     gsc_label("Input Channels");
264     errs  = dsi_query(fd, DSI_QUERY_CHANNEL_QTY, chans);
265     if (errs == 0)
266        printf("%ld Channels\n", (long) chans[0]);
267     return(errs);
268  }
```

A.1.3 fmcw_conf.c

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <fcntl.h>
4  #include <unistd.h>
5  #include <string.h>
6  #include <sys/time.h>
7  #include <ctype.h>
8  #include <math.h>
9  #include <errno.h>
10 #include <signal.h>
11 #include <stdint.h>
```

```
12 #include "fmcw_conf.h"
13
14 int read_fmcw_conf(const char cf_filename[], struct fmcw_conf_struct *conf) {
15
16   /* Open FMCW config file */
17   FILE *fp;
18   int line_ctr;
19   char current_line[256];
20   char *token_ptr;
21   char data_dir[256];
22   char desc[80];
23   float params[FMCW_CONF_NUM_PARAMS];
24   size_t param_check;
25
26   if ((fp = fopen(cf_filename, "r")) == NULL) {
27     fprintf(stderr, "fmcw: unable to open %s\n", cf_filename);
28     return FMCW_CONF_READ_ERROR;
29   }
30
31   for (line_ctr = 0; line_ctr < 10/*FMCW_CONF_NUM_PARAMS*/; line_ctr++) {
32
33     /* Read configuation file line */
34     if (fgets(current_line, sizeof current_line, fp) == NULL) {
35       fprintf(stderr, "fmcw: error reading from %s\n", cf_filename);
36       fclose(fp);
37       return FMCW_CONF_READ_ERROR;
38     }
39     // printf("%s:%d %s", cf_filename, line_ctr, current_line);
40     /* Search line for separator token ':' */
41     if ((token_ptr = strchr(current_line, ':')) == NULL) {
42       fprintf(stderr, "fmcw: separator token missing in %s:%d\n", cf_filename,
           line_ctr);
43       fclose(fp);
44       return FMCW_CONF_READ_ERROR;
45     }
46     /* Read the parameters */
47     /*switch (line_ctr) {
48     case DATA_DIR:
49       param_check = sscanf(token_ptr+sizeof(char), "%s", data_dir);
50        printf("fmcw: %s\n", data_dir);
51       if (param_check <= 0) {
52    fprintf(stderr, "fmcw: parameter incorrectly set in %s:%d\n", cf_filename,
        line_ctr);
53    fclose(fp);
54    return FMCW_CONF_READ_ERROR;
55       }
56       break;
57     default:*/
58       param_check = sscanf(token_ptr+1, "%f", &(params[line_ctr]));
59       if (param_check <= 0) {
60    fprintf(stderr, "fmcw: parameter incorrectly set in %s:%d\n", cf_filename,
        line_ctr);
61    fclose(fp);
62    return FMCW_CONF_READ_ERROR;
63       }
64        //printf("Params read %d %f\n", line_ctr, params[line_ctr]);
65       //break;
66     //}
67   }
68
69   fclose(fp);
70
71   /* Update configuration structure */
72   //strcpy(conf->data_dir, data_dir);
73   //printf("fmcw: data directory is %s\n", conf->data_dir);
74   //strcpy(conf->desc, desc);
75   //printf("Description: %s\n", conf->data_dir);
76   //conf->num_data_files = (int)(params[NUM_DATA_FILES]);
```

```
77    conf->ain_frame_scans = (int)(params[8]);
78    conf->bandwidth= (float) (params[1]);
79    printf("bandwidth=%f\n", conf->bandwidth);
80    conf->chirp_length= (float) (params[2]);
81    printf("Tp=%f\n", conf->chirp_length);
82    conf->sample_frequency= (long int)(params[4]);
83    printf("sample freq=%ld\n", conf->sample_frequency);
84    conf->prf= (int)(params[3]);//IGNORE
85    //printf("PRF=%d\n", conf->prf);
86    conf->samples_in_prof= (int)(conf->sample_frequency*conf->chirp_length-5);
87    printf("samples in prof=%d\n", conf->samples_in_prof);
88    conf->number_of_profiles= (int)(params[9]);
89    printf("Total Number of profiles=%d\n", conf->number_of_profiles);
90    return FMCW_NO_ERROR;
91
92  }
93
94  void fwrite_fmcw_conf(struct fmcw_conf_struct fmcw_conf,FILE* file)
95  {
96    int header_version=1;
97    fwrite(&header_version,sizeof(int),1,file);
98  int ain_frame_scans=10;//no matter
99    fwrite(&(ain_frame_scans),sizeof(int),1,file);
100 int num_channels=3;
101   fwrite(&(num_channels),sizeof(int),1,file);
102   fwrite(&(fmcw_conf.bandwidth),sizeof(float),1,file);
103   fwrite(&(fmcw_conf.chirp_length),sizeof(float),1,file);
104 int center_frequency=114.5;
105   fwrite(&(center_frequency),sizeof(int),1,file);
106   fwrite(&(fmcw_conf.sample_frequency),sizeof(long int),1,file);
107   fwrite(&(fmcw_conf.prf),sizeof(int),1,file);
108   fwrite(&(fmcw_conf.samples_in_prof),sizeof(int),1,file);
109   fwrite(&(fmcw_conf.number_of_profiles),sizeof(int),1,file);
110 }
```

For historical reasons the format of the configuration file `'fmcw2010.conf'` is the following:

```
Center Frequency (MHz) :114.5
Bandwidth (MHz) :25
Chirp Length(ms) :10
PRF (Hz) :100
Sample Frequency (KHz) :60
AO Sampling Frequency(kHz):40
Daq_trigger param :40000
Scans per Profile :595
Scans Per Frame :29750
Total Profiles :5000
Number of Channels :3
```

But there there are some parameters which currently are not used: `Daq_trigger param, Scans per frame` and `AO Sampling Frequency`.

B.1 Windprofiler2010

This folder contains all the required files for the signal processing and for the configuration of the FPGA. The files are briefly described below.

FPGA		
	`windprofiler_0409.c/exe`	Initializes the FPGA sending the parameters of the `.conf` file through the serial port.
	`reset.c/exe`	Resets the FPGA.
	`fmcw2010.conf`	Contains the parameters to configure the wind profiler.
	`fmcw_current.conf`	
Processing & Display		
	Data/	This folder contains all acquired data files from DAQ (`.dat`) and all processed files (`.fig`) .
	`windprof_process.pro`	Processes acquired data from `savedata_windprof`.
	`windprof_process2.pro`	It is already explained in Chapter 5.
	`windprof_process_realtime.pro`	Processes acquired data in real time.
	`windprof_display.pro`	Display processed data from `windprof_process.pro`.
	`windprof_display_realtime.pro`	Display processed data from `windprof_process_realtime.pro` in real time.
	`windprof_par2.pro`	Defines the header written by acquisition program to be read by `windprof_process.pro`.
	`windprof_par_out.pro`	Defines the header written by `windprof_process.pro` to be read by `windprof_display.pro.`
	`load_ct_rb.pro`	Functions required by `windprof_display.pro` and
	`imgunder.pro`	`windprof_display_realtime.pro.`
GUI		
	`wgui.pro`	Starts FMCW Profiler GUI.
	`wgui_create.pro`	Defines the graphical interface and buttons of the GUI.
	`wgui_event.pro`	Defines the functions of the buttons.
	`run_windprof.pro`	Executes `windprofiler_0409`. It loads drivers for DAQ and compiles savedata.
	`reset_windprof.pro`	Executes `reset.exe`.
	`run_daq.pro`	Executes `savedata`.
	`create_conf.pro`	Creates the `.conf` file.
	`run_process_realtime.pro`	Starts `windprof_process_realtime.pro` with the input
	`windprof.conf`	parameters that are contained in `windprof.conf`.
	`run_display.pro`	Starts `windprof_display.pro.`
	`run_display_realtime.pro`	Starts `windprof_display_realtime.pro.`

Table B.1: Windprofiler2010 files

B.1.1 Windprof_process.pro

```
; Carol Asturiano
; MIRSL Lab @ UMass Amherst
; Process acquired data from savedata_windprof
;
;               Steps for FMCW Processing
;
; 1. read header, creating a structure
; 2. read profile and convert data to time domain using fft
; 3. (pc=1)remove constant phase that appears in calibration signal
; 4. (mr=1)remove temporal mean to get rid of stationary targets such as
;    buildings and trees...
; 5. get power and average in time
; 6. get doppler using pulse-pair
;
; To execute: windprof_process, '02221042.dat',1,0,1,1,0

pro windprof_process, infile, time_avg, number_channel,mr,pc,delete_samples, png=png

  lun = 1             ; our input file
  lun_out = 2          ; our output file
  close, /all
  on_ioerror, at_eof  ;jump to at_eof in case of ioerror

  if (not keyword_set(delete_samples)) then delete_samples=0
  print,'mr=',mr,' pc=',pc,' delete_samples=',delete_samples

  lambda=3e8/915e6
  number_channels=1 ;4 number of channels on the file to process
  ch=number_channel

  infile = '/home/profiler/Desktop/windprofiler2010/data/' + infile
  outfile = infile + '.fig' ; that should be modified later for ordinary data
  print,"created file", outfile

  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;
  ;Read header, creating a structure
  ;
   print,"Reading header ..."
   openr, lun, infile
   openw, lun_out, outfile

   header_version=1L
   readu,lun,header_version
   if header_version ne 1 then begin
      print,"Wrong header version"
   ;   return
   endif
   print, " header version:", header_version

   par = windprof_par2()
   readu, lun, par
   print,"  ain_frame_scans",par.ain_frame_scans
   print,"  ain_num_channels",par.ain_num_channels
   print,"  bandwidth",par.bandwidth
   print,"  chirp_length",par.chirp_length
      print,"  center_frequency",par.center_frequency
   print,"  sample_freq",par.sample_freq
   print,"  prf",par.prf
   print,"  scans_in_prof",par.scans_in_prof
   print,"  total number profiles",par.total_number_profiles

   Block_size=par.scans_in_prof
   prf=par.prf
```

```
65      info=fstat(lun)
66      tam=info.size
67      tamheader=10
68      print, "File size:", tam, " bytes"
69      number_samples=tam/4-tamheader ;Total samples, (4 bytes/sample)
70      number_profiles=number_samples/Block_size
71      print, "Calculated Number of profiles:",number_profiles
72      number_profiles=par.total_number_profiles
73
74  ;Create output header...
75  par_out = windprof_par_out()
76
77  par_out.bandwidth=par.bandwidth
78  par_out.chirp_length=par.chirp_length
79  par_out.center_frequency=par.center_frequency
80  par_out.sample_frequency=par.sample_freq
81  par_out.prf=par.prf
82  par_out.scans_in_prof=(Block_size-delete_samples)/2
83  par_out.time_avg=time_avg
84  prof_avg=time_avg*par.prf
85  par_out.number_of_profiles=number_profiles/prof_avg
86
87  print,"par out: scans in prof",par_out.scans_in_prof
88  ;print,"par out", par_out
89  ;write it to the file
90  writeu,lun_out,par_out
91
92    ;Data should be transformed to real Voltage values.
93    ;1LSB equals the full escale (20V) divided by 1.1921uV (24 bits)
94    LSB = 20D/(2D^24D)
95    ;the first 8 bits are discarded, they indicated the channel only
96    mask=16777215 ;4278190080 = FF000000 and 16777215=00FFFFFF
97
98  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
99  print, 'Processing ...   Prof. avg=',prof_avg
100
101 f_frame=dblarr(Block_size-delete_samples, prof_avg)
102 raw_data =lonarr(number_channels,Block_size)
103 ch_info=dblarr(number_channels,Block_size)
104
105 wsamples=Block_size-delete_samples
106 windw = rebin(hanning(wsamples), wsamples, prof_avg)
107 winfunc=dblarr(Block_size-delete_samples,prof_avg)
108 winfunc[0:wsamples-1,*]=windw
109
110 for avg_frame_index=0L, number_profiles/prof_avg-1  do begin ;big loop
111
112    for frame_index= 0L, prof_avg-1 do begin
113    readu, lun, raw_data
114    raw_data=mask AND raw_data;the first 8 bits are discarded, they indicated the
           channel only
115    ch_info=raw_data*LSB-10;transforming integer to real voltage value
116    f_frame[*,frame_index] = ch_info(ch,delete_samples:Block_size-1)
117       ;print,'f_frame_index',frame_index
118    endfor
119
120   f_frame=f_frame*winfunc
121   c_frame=(fft(f_frame,dimension=1))[0:(Block_size-delete_samples)/2-1,*];
122
123    if(pc eq 1) then begin
124     ;Do phase correction
125     j=complex(0,1)
126     fb=2.0*par.bandwidth/par.chirp_length
127     fs=par.sample_freq
128     i_ref=round(fb*(Block_size+5-delete_samples)/fs); it corresponds to fb (=5kHz)
           when delete_samples=0 and default config;
129     fi_ref=atan(conj(c_frame[i_ref,*]),/phase)
130     for i= 0, (Block_size-delete_samples)/2-1 do begin
```

```
131         c_cor=exp(j*fi_ref*i/i_ref)
132       c_frame[i,*]=c_frame[i,*]*c_cor
133        endfor
134        ;print,'Block',Block_size,'fs',fs,'fb cal',fb,' i ref', i_ref
135       endif
136
137        c_mean=total(c_frame,2)/prof_avg
138
139        if(mr eq 1) then begin
140        c_frame1=c_frame-complex(rebin(real_part(c_mean),(Block_size-delete_samples)/2,
               prof_avg),rebin(imaginary(c_mean),(Block_size-delete_samples)/2, prof_avg))
141        endif else begin
142        c_frame1=c_frame
143        endelse
144
145        pow=total(double(c_frame*conj(c_frame)), 2)/prof_avg
146        pow1=(total(double(c_frame1*conj(c_frame1)), 2)/prof_avg)
147
148        autocorr=total(c_frame[*,1:prof_avg-1]* $
149                       conj(c_frame[*,0:prof_avg-2]),2) / (prof_avg-1)
150        autocorr_phase=atan(autocorr,/phase)
151
152        autocorr1=total(c_frame1[*,1:prof_avg-1]* $
153                       conj(c_frame1[*,0:prof_avg-2]),2) / (prof_avg-1)
154        autocorr_phase1=atan(autocorr1,/phase)
155
156        ;Get velocity
157        ;vel=autocorr_phase*(prf*lambda)/(4*!pi)
158        vel1=autocorr_phase1*(prf*lambda)/(4*!pi)
159
160        ;Get coherence
161        ;rho=abs(autocorr)/pow
162        rho=abs(autocorr1)
163        rho1=abs(autocorr1)/pow1
164
165         ;powdb=10*alog10(pow)
166         ;powdb1=10*alog10(pow1)
167
168       ;writeu,lun_out,pow,vel,rho,c_mean
169       writeu,lun_out,pow1,vel1,rho1,c_mean
170
171   endfor
172
173   print,'Amplitude max ch_info',max(ch_info[ch,*]),'V   min ch_info',min(ch_info[ch,*])
          ,'V'
174   ;print,'Pw max',max(pow1),'W =',10*alog10(max(pow1))+30,'dBm   min',min(pow1),'W'
175
176   titlestr1 = 'Chirp Amplitude  ' + strmid(infile, 45, 13)
177   titlestr2 = 'Signal Power at receiver input ' + strmid(infile, 45, 13)
178
179   window, XSize=600, YSize=600
180   !p.multi=[0,1,2]
181   plot, ch_info(ch,delete_samples:Block_size-1), title=titlestr1,xtitle='samples'
182   if (delete_samples ne 0) then begin
183        str='Delete samples:'+strtrim(delete_samples,2)
184       xyouts,450,8,str
185   endif
186
187   ;oplot,ch_info, COLOR='FF00FF'x
188
189   Grec=-114.5+10
190   plot, 10*alog10(pow1)+30+Grec,title=titlestr2,xtitle='samples',ytitle='dBm',YRANGE
          =[-200,-80]
191   ;Pmax=10*alog10(max(pow1))+30+Grec
192
193   str='Profiles average:'+strtrim(prof_avg,2)
194   xyouts,(Block_size-delete_samples)/2-75,-90,str
195
```

```
;noise-floor
oplot, 10*alog10(rho)+30+Grec,COLOR='00FF00'x


at_eof:

close,lun
close,lun_out
print, 'File created.'

;Save plot in a .png file if option is set
if keyword_set(png) then begin
   a=tvrd(/true)
   write_png, infile + '.png',a
endif

end
```

B.1.2 Windprof_process2.pro

The only difference with the previous one, is that after the phase correction makes the sum of two consecutive chirps.

```
; add 2 consecutive chirps
  k=0U
  for j= 0,(prof_avg/2 -1) do begin
   c_frame2[*,j]=c_frame[*,k]+c_frame[*,k+1]
   k=k+2
```

B.1.3 Windprof_display.pro

```
; Carol Asturiano
; MIRSL Lab @ UMass Amherst
;
;  Display the processed data file
;
; windprof_display, '02011520.dat.fig', pixmap=pixmap, png=1

pro windprof_display, file, pixmap=pixmap, png=png
    device,decomposed=0
    ; Physical constants
    c = 299792458.0 ; m/s
   par_out = windprof_par_out()

   lun_out = 2
   close, lun_out
   file = '/home/profiler/Desktop/windprofiler2010/data/' + file
   openr, lun_out, file
   readu, lun_out, par_out
   print, par_out

   range_bins=par_out.scans_in_prof
   print,"range bins",par_out.scans_in_prof

;
; calculate display range resolution
;
   deltaR = c/(2e6*par_out.bandwidth)
   print,"deltaR",deltaR
```

```
29
30      ;r = deltaR*findgen(range_bins)* par_out.prf* par_out.chirp_length
31      ; To accound for the delay that we put in the LO of the mixer(reference chirp) we
            need to add 15m to each range bin
32      r = deltaR*findgen(range_bins)* par_out.prf* par_out.chirp_length*1e-3
33      ;r(0) = 1.0 ; Don't multiply the first range gate by zero
34
35  ; Variables
36       profs = par_out.number_of_profiles
37       print,"number of profs",profs
38
39  ;
40  ; declare processed data records in .img file
41  ;
42       pwr = dblarr(range_bins) ; Single profile
43       velocity = dblarr(range_bins)
44       rho = dblarr(range_bins)
45       avg = dcomplexarr(range_bins)
46
47  ;
48  ; declare 2D (range-time image) for power, vel, rho, etc.
49  ;
50       rti = dblarr(profs, range_bins)
51       rtv = dblarr(profs, range_bins)
52       rtrho = dblarr(profs, range_bins)
53       rtc = dblarr(profs, range_bins)
54
55  ;
56  ;  jump to at_eof: if we hit end-of-file
57  ;
58       on_ioerror, at_eof
59       for i = 0L, profs-1 do begin
60            readu,lun_out,pwr,velocity,rho,avg
61           rti[i,*] = pwr
62           rtv(i,*) = velocity
63           rtrho(i,*) = rho
64           rtc(i,*) = double(avg*conj(avg))
65       endfor
66
67  at_eof:
68       if (i lt profs-1) then begin
69         profs = i-1
70          rti = rti(0:profs-1,*)
71         rtv = rtv(0:profs-1,*)
72        rtrho = rtrho(0:profs-1,*)
73          rtc = rtc(0:profs-1,*)
74       endif
75
76       close, lun_out
77       free_lun, lun_out
78
79  ;
80  ; create time vector (time in minutes starting at zero)
81  ;
82       time_avg=par_out.time_avg
83       t=findgen(profs)
84       t=t * time_avg/60
85  ;
86  ; R^2 range correction . Note: Here the filter freq response should also be take into
        account later!!!
87  ;
88      r2 = rebin(transpose(r^2)/1e6, profs, range_bins)
89      rti=rti*r2
90     ;rti = rti*rtrho ;rti*r2*rtrho
91     ; mask = rtrho
92     ; mask(*) = 1
93     ; idx = where(rtrho lt 0.3)
94     ; mask(idx) = 0
```

```
95    ;rtv = rtv*mask
96
97  window, 0, xsize=810, ysize=600;, pixmap=pixmap
98      !p.multi=[0,1,2]
99      print,'Pw max',max(rti),'W   min',min(rti),'W'
100
101     loadct, 5
102     titlestr = 'UMass FMCW:  ' + strmid(file, 45, 16)
103     contour, rti, t, r, /nodata, /xs, /ys, title=titlestr, xtitle='time (min)',
            ytitle='height (m)';Color=!P.Background,Background=!P.Color
104     imgunder, bytscl(10*alog10(rti)+30, min=-40, max=0)
105     contour, rti, t, r, /nodata, /xs, /ys, /noerase, title=titlestr, xtitle='time (
            min)', ytitle='height (m)';Color=!P.Background,Background=!P.Color
106
107     loadct_rb
108     contour, rtv, t, r, /nodata, /xs, /ys
109     imgunder, bytscl(rtv);, min=-1,max=1)
110     contour, rtv, t, r, /nodata, /xs, /ys, /noerase
111     !p.multi=0
112
113     if keyword_set(png) then begin
114        a = tvrd(/true)
115        write_png, file + '.png', a
116     endif
117 stop
118
119 end
```

B.1.4 Windprof_process_realtime.pro

```
1  ; Carol Asturiano
2  ; MIRSL Lab @ UMass Amherst
3  ;
4  ; Process acquired data from savedata_windprof
5  ;
6  ; 1. read header, creating a structure
7  ; 2. read profile and convert data to time domain using fft
8  ; 3. perform phase correction
9  ; 4. remove temporal mean to get rid of stationary targets such as
10 ;    buildings and trees..
11 ; 5. get power and average in time
12 ; 6. get doppler using pulse-pair
13 ;
14 ; windprof_process_realtime, '02091045.dat',1,0
15
16 pro windprof_process_realtime, infile,time_avg, number_channel
17
18   lun = 1              ; our input file
19   lun_out = 2           ; our output file
20   close, /all
21   on_ioerror, at_eof  ;jump to at_eof in case of ioerror
22
23   lambda=3e8/915e6
24   number_channels=1 ;4 number of channels on the file to process
25   ch=number_channel
26
27   infile = '/home/profiler/Desktop/windprofiler2010/data/' + infile
28   outfile = infile + '.fig' ; that should be modified later for ordinary data
29   print,"created file", outfile
30
31 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
32 ;
33 ;Read header, creating a structure
34 ;
35
```

```
36     print,"Reading header ..."
37     openr, lun, infile
38     openw, lun_out, outfile
39     header_version=1L
40
41  ;
42  ;
43  ; wait for header to be written to file
44  ;
45  ;
46      header_size=11
47      fs = fstat(lun)
48      while (fs.size le header_size) do begin ;specify header lenght!!!!!!!!!!!!!!!!!
49      wait, 1
50      fs = fstat(lun)
51      endwhile
52
53  readu,lun,header_version
54
55  print, 'header read'
56  if header_version ne 1 then begin
57      print,"Wrong header version"
58      return
59  endif
60  print, " header version:", header_version
61
62  par = windprof_par2()
63     readu, lun, par
64     print,"  ain_frame_scans",par.ain_frame_scans
65     print,"  ain_num_channels",par.ain_num_channels
66     print,"  bandwidth",par.bandwidth
67     print,"  chirp_length",par.chirp_length
68      print," center_frequency",par.center_frequency
69     print,"  sample_freq",par.sample_freq
70     print,"  prf",par.prf
71     print,"  scans_in_prof",par.scans_in_prof
72     print,"  total number profiles",par.total_number_profiles
73
74     Block_size=par.scans_in_prof
75     prf=par.prf
76     info=fstat(lun)
77     tam=info.size
78     tamheader=10
79     print, "File size:", tam, " bytes"
80     number_samples=tam/4-tamheader ;Total samples, (4 bytes/sample)
81          number_profiles=par.total_number_profiles
82     number_profiles2=number_samples/Block_size
83     print, "Calculated Number of profiles:",number_profiles2
84  ;stop
85
86  ;Create output header...
87  par_out = windprof_par_out()
88
89  par_out.bandwidth=par.bandwidth
90  par_out.chirp_length=par.chirp_length
91  par_out.center_frequency=par.center_frequency
92  par_out.sample_frequency=par.sample_freq
93  par_out.prf=par.prf
94  par_out.scans_in_prof=(Block_size)/2
95  par_out.time_avg=time_avg
96
97  prof_avg=time_avg*par.prf
98  print,'prof avg',prof_avg
99
100 par_out.number_of_profiles=number_profiles/prof_avg
101 ;print,"par out", par_out
102
103 ;write it to the file
```

```
104 writeu,lun_out,par_out
105
106   ;Data should be transformed to real Voltage values.
107   ;1LSB equals the full escale (20V) divided by 1.1921uV (24 bits)
108   LSB = 20D/(2D^24D)
109   ;the first 8 bits are discarded, they indicated the channel only
110   mask=16777215 ;4278190080 = FF000000 and 16777215=00FFFFFF
111 ;;;
```

```
    fb=2.0*par.bandwidth/par.chirp_length
    fs=par.sample_freq
    i_ref=round(fb*(Block_size+5)/fs); it corresponds to fb (=5kHz, Default conf.);
    fi_ref=atan(conj(c_frame[i_ref,*]),/phase)
    for i= 0, (Block_size)/2-1 do begin
      c_cor=exp(j*fi_ref*i/i_ref)
       c_frame[i,*]=c_frame[i,*]*c_cor
    endfor

    c_mean=total(c_frame,2)/prof_avg
    c_frame1=c_frame-complex(rebin(real_part(c_mean),Block_size/2, prof_avg),rebin(
        imaginary(c_mean),Block_size/2, prof_avg))

    pow=total(double(c_frame*conj(c_frame)), 2)/prof_avg
    pow1=(total(double(c_frame1*conj(c_frame1)), 2)/prof_avg)

    autocorr=total(c_frame[*,1:prof_avg-1]* $
                   conj(c_frame[*,0:prof_avg-2]),2) / (prof_avg-1)
    autocorr_phase=atan(autocorr,/phase)

    autocorr1=total(c_frame1[*,1:prof_avg-1]* $
                   conj(c_frame1[*,0:prof_avg-2]),2) / (prof_avg-1)
    autocorr_phase1=atan(autocorr1,/phase)

    ;Get velocity
    vel=autocorr_phase*(prf*lambda)/(4*!pi)
    vel1=autocorr_phase1*(prf*lambda)/(4*!pi)

    ;Get coherence
    rho=abs(autocorr)/pow
    rho1=abs(autocorr1)/pow1

     ;powdb=10*alog10(pow)
     ;powdb1=10*alog10(pow1)
   ;writeu,lun_out,pow,vel,rho,c_mean
   writeu,lun_out,pow1,vel1,rho1,c_mean
endfor

at_eof:
close,lun
close,lun_out
print, 'File created.'

end
```

B.1.5 Windprof_display_realtime.pro

```
; Carol Asturiano
; MIRSL Lab @ UMass Amherst
;
;  Display the processed data file in real time
;
; windprof_display_realtime, '02091310.dat.fig', pixmap=pixmap, png=png

pro windprof_display_realtime, file, pixmap=pixmap, png=png


    if (not keyword_set(pixmap)) then pixmap=0

    ; Physical constants
    c = 299792458.0               ; m/s
    profile_size= 297*5*8      ;define the size of a pow1, vel, rho, c_mean block
;
; create plotting window on the screen
;
```

```
19    wid=10
20    window, wid,title='UMass FMCW: (Right click to Stop or Left click to freeze)',
          xsize=800, ysize=600, pixmap=pixmap
21  ;
22  ; Create plotting window on memory (pixmap), we'll draw on this one
23  ;
24
25    pix=11
26    window, pix, xsize=800, ysize=600, /pixmap
27
28     par_out = windprof_par_out()
29
30
31     lun_out = 2
32     close, lun_out
33     file = '/home/profiler/Desktop/windprofiler2010/data/' + file
34     openr, lun_out, file
35     readu, lun_out, par_out
36     print, par_out
37
38     range_bins=par_out.scans_in_prof
39     print,"range bins",par_out.scans_in_prof
40
41
42  ;
43  ; calculate display range resolution
44  ;
45     deltaR = c/(2e6*par_out.bandwidth)
46     print,"deltaR",deltaR
47
48     r = deltaR*findgen(range_bins)* par_out.prf* par_out.chirp_length*1e-3
49
50  ; Variables
51  ;
52      profiles = par_out.number_of_profiles
53      time_index = indgen(par_out.number_of_profiles)
54      profs = profiles
55      print,"number of profs",profs
56
57
58  ;
59  ; declare processed data records in .img file
60  ;
61      pwr = dblarr(range_bins) ; Single profile
62      velocity = dblarr(range_bins)
63      rho = dblarr(range_bins)
64      avg = dcomplexarr(range_bins)
65
66  ;
67  ; declare 2D (range-time image) for power, velocity, rho, etc.
68  ;
69      rti = dblarr(profs, range_bins)
70      rtv = dblarr(profs, range_bins)
71      rtrho = dblarr(profs, range_bins)
72      rtc = dblarr(profs, range_bins)
73
74
75  ;
76  ;  jump to at_eof: if we hit end-of-file
77  ;
78   on_ioerror, at_eof
79
80
81      read_ok=0 ;create the var read_ok to make sure in every loop that there's data
            available
82      prof=0 ; count the number of read profiles so far
83      change =0 ; variable that says if we've read new data, so we've to refresh the
            window
```

```
84      for i = 0L, profiles-1 do begin ;insert here the wait loop
85
86     read_profiles:
87          ;print, 'read profiles, readok',read_ok,'change',change,' i', i,' prof',
                profiles-1
88          wset, pix ; current window equal pixmap!!!
89          ;
90          ;
91          while (read_ok eq 0) do begin ;we'll wait till there's a new profile
                available to read
92            info=fstat(lun_out) ;get the read file status
93            current=info.cur_ptr
94            fsize=info.size
95            pos_to_eof=fsize-current
96            ;print,'pointer: ', current, 'pos_to_eof: ', pos_to_eof
97            if (pos_to_eof gt profile_size) then begin
98               read_ok=1  ; we have enough data to read
99               change=1 ;we're goin' to read more data, so we'll need to refresh the
                     window
100           endif else begin
101              read_ok=0
102              wait, 0.2 ;we wait for the data to be ready
103           if(change eq 1) then begin
104              goto, display ;we display data while the data is not ready
105           endif
106           endelse
107
108           wset, wid
109           cursor, ix, iy, /nowait
110            if(!mouse.button eq 4) then goto, at_eof2 $ ;right click, exit
111           else if(!mouse.button eq 1) then $
112            cursor, ix, iy, /up ;left button, freeze
113            wset, pix
114         endwhile
115         ;
116         ;
117         ;
118
119         ;info=fstat(lun_out)
120    ;print, 'curent pointer before read:   ', info.cur_ptr
121
122
123         readu,lun_out,pwr,velocity,rho,avg
124
125         rti[i,*] = pwr
126         rtv(i,*) = velocity
127         rtrho(i,*) = rho
128         rtc(i,*) = double(avg*conj(avg))
129
130         ;info=fstat(lun_out)
131    ;print, 'curent pointer after read:   ', info.cur_ptr
132
133         prof=prof +1 ; we've read one more profile
134         ;print,' prof#',prof
135         read_ok=0
136     endfor
137
138 ;stop
139
140 at_eof:
141
142     if (i lt profs-1) then begin ;if ioerror
143        profs = i-1
144        rti = rti(0:profs-1,*)
145        rtv = rtv(0:profs-1,*)
146        rtrho = rtrho(0:profs-1,*)
147        rtc = rtc(0:profs-1,*)
148     endif
```

```
149
150      close, lun_out
151      free_lun, lun_out
152
153  display:  ;if there's no data available we'll display the data we have in the
         meantime
154  ;print,' display..'
155
156  ;
157  ; create time vector (time in minutes starting at zero)
158  ;
159      time_avg=par_out.time_avg
160      t=findgen(profs)
161      t=t * time_avg/60
162
163  ;
164  ; R^2 range correction
165  ;
166      r2 = rebin(transpose(r^2)/1e6, profs, range_bins)
167     ;rti=rti*r2
168     ;rti = rti*rtrho ;rti*r2*rtrho
169     ;mask = rtrho
170     ;mask(*) = 1
171     ;idx = where(rtrho lt 0.3)
172     ;mask(idx) = 0
173     ;rtv = rtv*mask
174
175      ;pfact = profs/900
176      ;rti_disp = rebin(rti, profs/pfact, range_bins)
177      ;rtv_disp = rebin(rtv, profs/pfact, range_bins)
178      ;rtc_disp=rebin(rtc, profs/pfact, range_bins)
179      ;t = rebin(t, profs/pfact)
180
181      ;stop
182
183      !p.multi=[0,1,2]
184
185      loadct, 5
186      titlestr = 'UMass FMCW:  ' + strmid(file, 45, 16)
187      contour, rti, t, r, /nodata, /xs, /ys, title=titlestr, xtitle='time (min)',
             ytitle='height (m)';Color=!P.Background,Background=!P.Color
188      imgunder, bytscl(10*alog10(rti)+30, min=-40, max=-10)
189      contour, rti, t, r, /nodata, /xs, /ys, /noerase, title=titlestr, xtitle='time (
             min)', ytitle='height (m)';Color=!P.Background,Background=!P.Color
190
191
192      loadct_rb
193      contour, rtv, t, r, /nodata, /xs, /ys
194      imgunder, bytscl(rtv);, min=-1,max=1)
195      contour, rtv, t, r, /nodata, /xs, /ys, /noerase
196
197
198      !p.multi=0
199
200  ;
201  ; Now copy the pixmap to the display window
202  ;
203      wset, wid
204      if(change eq 1) then begin
205      device, copy=[0, 0, !d.x_vsize, !d.y_vsize,0 ,0 , pix]
206      endif
207
208      change=0 ;we've refresh the window so we wait to the next read data
209
210      if keyword_set(png) then begin
211         a = tvrd(/true)
212         write_png, strmid(file, 0, 14) + '.png', a
213      endif
```

```
214 ;
215 ;Check to see if a mouse button has been pressed
216 ;
217
218 cursor, ix, iy, /nowait
219 if(!mouse.button eq 4) then goto, at_eof2 $ ;right click, exit
220
221 else if(!mouse.button eq 1) then $
222 cursor, ix, iy, /up ;left button, freeze
223 ;
224 ;If we have more profiles to read let's go and try to read them!
225 ;
226     if (prof lt par_out.number_of_profiles) then begin
227     goto, read_profiles
228     endif
229 ;stop
230
231 at_eof2:
232     close, lun_out
233     free_lun, lun_out
234     wdelete,pix
235     wdelete, wid
236     return
237 end
```

B.1.6 GUI

B.1.7 wgui_create.pro

```
1  ; Carol Asturiano
2  ; MIRSL Lab @ UMass Amherst
3  ;
4  ; for UHF-FMCW Radar
5  ;
6
7  PRO wgui_create
8
9  COMMON SHARE1, hdr, display, datafile, lun
10
11   display=1
12   datafile='test.dat'
13   ;hdr = header_w()
14
15    base = widget_base(title='UMass FMCW Radar GUI v2.0', /row)
16 ;
17 ;  Positioner settings
18 ;
19 b0 = widget_base(base, /frame, /column)
20 t0 = widget_label(b0, value="-- Radar Settings --")
21
22 b1 = widget_base(b0, /row)
23 t2 = widget_label(b1, value='   Bandwidth:          ')
24 t2 = widget_text(b1, /editable, xsize=8, uname='bandwidth', value='25')
25 t2 = widget_label(b1, value='[MHz]')
26
27 b1 = widget_base(b0, /row)
28 t2 = widget_label(b1, value='   Tp:                 ')
29 t2 = widget_text(b1, /editable, xsize=8, uname='Tp', value='10')
30 t2 = widget_label(b1, value='[ms]')
31
32 b1 = widget_base(b0, /row)
33 t2 = widget_label(b1, value='   100% duty cycle')
34 b1 = widget_base(b0, /row)
35 t2 = widget_label(b1, value='   Center Frequency: 915  [MHz]')
```

```
36
37  b1 = widget_base(b0, /row)
38  t2 = widget_label(b1, value='    ')
39
40  b1 = widget_base(b0, /row)
41  t0 = widget_label(b0, value="-- Acquisition Settings --")
42
43  b1 = widget_base(b0, /row)
44  t2 = widget_label(b1, value='   Sample Frequency:  ')
45  t2 = widget_text(b1, /editable, xsize=8, uname='sample_freq', value='60')
46  t2 = widget_label(b1, value='[kHz]')
47
48  b1 = widget_base(b0, /row)
49  t2 = widget_label(b1, value='   Number of profiles:')
50  t2 = widget_text(b1, /editable, xsize=8, uname='number_profiles', value='1000')
51
52  b1 = widget_base(b0, /row)
53
54  ;
55  ;  Radar buttons
56  ;
57
58  b1 = widget_base(base, /frame, /column)
59  t0 = widget_label(b1, value='-- Radar configuration --')
60
61  b2 = widget_button(b1, value='Start Radar', uname="start_radar",uvalue="start_radar")
62  b2 = widget_button(b1, value='Reset Radar', uname="reset_radar",uvalue="reset_radar")
63
64  ;b2 = widget_base(b1, /row)
65  ;t2 = widget_label(b2, value='    ')
66
67  t2 = widget_label(b1, value="-- Data collection --")
68
69  b2 = widget_button(b1, value='Start Acquisition', uname="data_start", uvalue="
        data_start")
70  b2 = widget_button(b1, value='Stop Acquisition', uname="data_stop", uvalue="data_stop
        ")
71
72  t2 = widget_label(b1, value="-- Display Data --")
73
74  b2 = widget_button(b1, value='Process Data', uname="process_data", uvalue="
        process_data")
75  b2 = widget_button(b1, value='Process-Adding 2 chirps', uname="process_data2", uvalue
        ="process_data2")
76
77  b2 = widget_button(b1, value='Display Data', uname="display_data", uvalue="
        display_data")
78
79  b2 = widget_base(b1, /row)
80  ;t0 = widget_label(b2, value="Current file:")
81  ;b2 = widget_base(b1, /row)
82  ;t2 = widget_text(b2, xsize=24, uname='data_file', value='data.dat')
83
84  b2 = widget_button(b1, value='Process Real Time', uname="process_realtime",uvalue="
        process_realtime")
85
86  b2 = widget_button(b1, value='Display Real Time', uname="display_realtime",uvalue="
        display_realtime")
87
88  b3 = widget_base(base, /frame, /column)
89  t0 = widget_label(b3, value='-- Processing Options--')
90
91  b4 = widget_base(b3, /row)
92  t2 = widget_label(b4, value='Number channel: ')
93  t2 = widget_combobox(b4, uname='number_chan', value=['0', '1', '2'], uvalue='
        number_chan')
94
95  b4 = widget_base(b3, /row)
```

```
96  t2 = widget_label(b4, value='Time average:    ')
97  t2 = widget_text(b4, /editable, xsize=8, uname='time_average', value='1')
98  t2 = widget_label(b4, value='[s]')
99
100 b4 = widget_base(b3, /row)
101 t2 = widget_label(b4, value='Mean Removal:    ')
102 t2 = widget_combobox(b4, uname='mean_removal', value=['Yes', 'No'], uvalue='
        mean_removal')
103
104 b4 = widget_base(b3, /row)
105 t2 = widget_label(b4, value='Phase Correction:')
106 t2 = widget_combobox(b4, uname='phase_correction', value=['Yes', 'No'], uvalue='
        phase_correction')
107
108 b4 = widget_base(b3, /row)
109 t2 = widget_label(b4, value='Delete Samples:   ')
110 t2 = widget_text(b4, /editable, xsize=8, uname='delete_samples', value='0')
111 b4 = widget_base(b3, /row)
112 t2 = widget_label(b4, value='                  (usually 80)')
113
114 b4 = widget_base(b3, /row)
115 t2 = widget_label(b4, value='Save .png ?      ')
116 t2 = widget_combobox(b4, uname='save_png', value=['No','Yes'], uvalue='save_png')
117
118 setenv, 'INIT=1'
119
120 widget_control, base, /realize
121 xmanager, 'wgui', base
122
123 END
```

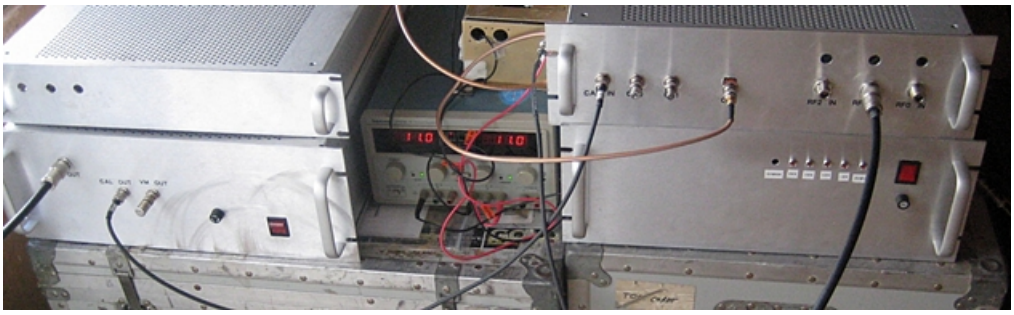
B.1.8 wgui_event.pro

```
1   ; Carol Asturiano
2   ; MIRSL Lab @ UMass Amherst
3   ;
4   ; for UHF-FMCW Radar
5   ;
6
7   PRO wgui_event, ev
8
9   COMMON SHARE1, hdr, display, datafile, lun
10
11  widget_control, ev.id, get_uvalue=value
12
13  case (value) of
14
15     "reset_radar": begin
16     reset_windprof
17     print, ".................radar reset done"
18     return
19     end
20
21    "start_radar": begin
22     print, 'radar start'
23     create_conf, ev ; create the .conf file
24     run_windprof
25     print, ".................radar start done"
26     return
27     end
28
29     "data_start": begin
30     print, 'data acquisition start'
31     create_conf, ev; create the .conf file
32     run_daq  ; inits data acquisition
```

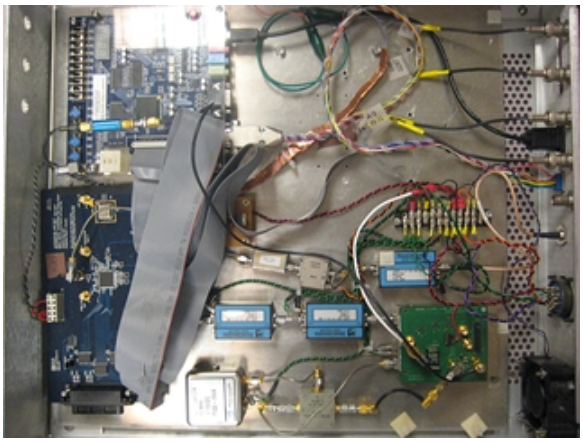
```
33      return
34      end
35
36      "data_stop": begin
37      print, 'data acquisition stop!'
38      spawn, 'sudo killall -s TERM savedata'
39      return
40      end
41
42      "process_data": begin
43      print, '.................start processing data'
44      id= widget_info(ev.top, find_by_uname='number_chan')
45      num=widget_info(id, /combobox_gettext)
46      number_channel=fix(num)
47
48      id = widget_info(ev.top, find_by_uname='time_average')
49      widget_control, id, get_value=time_average
50      time_avg=fix(time_average[0])
51
52      id = widget_info(ev.top, find_by_uname='delete_samples')
53      widget_control, id, get_value=delete_samples
54      delete_samples=fix(delete_samples[0])
55
56      id= widget_info(ev.top, find_by_uname='mean_removal')
57      mr1=widget_info(id, /combobox_gettext)
58      mr=strcmp(mr1,'Yes')
59      print,'mean removal=',mr1
60
61      id= widget_info(ev.top, find_by_uname='phase_correction')
62      pc1=widget_info(id, /combobox_gettext)
63      pc=strcmp(pc1,'Yes')
64      print,'phase correction=',pc1
65
66      id= widget_info(ev.top, find_by_uname='save_png')
67      pn1=widget_info(id, /combobox_gettext)
68      png1=strcmp(pn1,'Yes')
69      print,'save png=',png1
70
71      print,'o Time average:', time_average
72      print,'o Number channel:',number_channel
73      print,'o Delete samples:',delete_samples
74
75      file=pickfile(/READ, PATH='/home/profiler/Desktop/windprofiler2010/data',FILTER ='
            *.dat')
76      infile=strmid(file, 45, 13)
77      print,"file name:  ",infile
78      windprof_process, infile, time_avg, number_channel,mr,pc, delete_samples, png=png1
79      print, '.................processing done'
80      return
81      end
82
83    "process_data2": begin
84      print, '.................start processing data'
85      id= widget_info(ev.top, find_by_uname='number_chan')
86      num=widget_info(id, /combobox_gettext)
87      number_channel=fix(num)
88
89      id = widget_info(ev.top, find_by_uname='time_average')
90      widget_control, id, get_value=time_average
91      time_avg=fix(time_average[0])
92
93      id = widget_info(ev.top, find_by_uname='delete_samples')
94      widget_control, id, get_value=delete_samples
95      delete_samples=fix(delete_samples[0])
96
97      id= widget_info(ev.top, find_by_uname='mean_removal')
98      mr1=widget_info(id, /combobox_gettext)
99      mr=strcmp(mr1,'Yes')
```

```
100     print,'mean removal=',mr1
101
102     id= widget_info(ev.top, find_by_uname='phase_correction')
103     pc1=widget_info(id, /combobox_gettext)
104     pc=strcmp(pc1,'Yes')
105     print,'phase correction=',pc1
106
107     id= widget_info(ev.top, find_by_uname='save_png')
108     pn1=widget_info(id, /combobox_gettext)
109     png1=strcmp(pn1,'Yes')
110     print,'save png=',png1
111
112     print,'o Time average:', time_average
113     print,'o Number channel:',number_channel
114     print,'o Delete samples:',delete_samples
115
116     file=pickfile(/READ, PATH='/home/profiler/Desktop/windprofiler2010/data',FILTER ='
            *.dat')
117     infile=strmid(file, 45, 13)
118     print,"file name:  ",infile
119     windprof_process2, infile, time_avg, number_channel,mr,pc, delete_samples, png=
            png1
120     print, '.................processing done'
121     return
122     end
123
124     "process_realtime":begin
125     print, '.................start processing real time'
126     id= widget_info(ev.top, find_by_uname='number_chan')
127     num=widget_info(id, /combobox_gettext)
128     number_channel=fix(num)
129     id = widget_info(ev.top, find_by_uname='time_average')
130     widget_control, id, get_value=time_average
131     time_avg=fix(time_average[0])
132     print,'o Time average:', time_average
133     print,'o Number channel:',number_channel
134     ls= file_search('data/*.dat') ;we search all the .fig files on a folder, and store
            the result on an array
135       x= size(ls) ;we figure out the size of the array
136       x=x(1)-1 ; find the number of elements in ls, and then we adquire the last one
137       file_name=ls(x)
138       file_name=strmid(file_name, 5, 19)
139     ;id = widget_info(ev.top, find_by_uname='data_file')
140     ;widget_control, id, set_value=file_name
141     print, 'the name of the last .dat file is: ',file_name
142     lun_proc=13
143     close, lun_proc
144     close, 14
145     openw, 14, 'windprof.conf'
146     printf, 14, file_name
147     printf, 14, time_avg
148     printf, 14, number_channel
149     close, 14
150     spawn, "idl run_process_realtime", unit=lun_proc
151     return
152     end
153
154     "display_data": begin
155     lun_disp=12
156     close, lun_disp
157     spawn, "idl run_display", unit=lun_disp
158
159     end
160
161     "display_realtime": begin
162     ls= file_search('data/*.fig') ;we search all the .fig files on a folder, and store
            the result on an array
163     x= size(ls) ;we figure out the size of the array
```

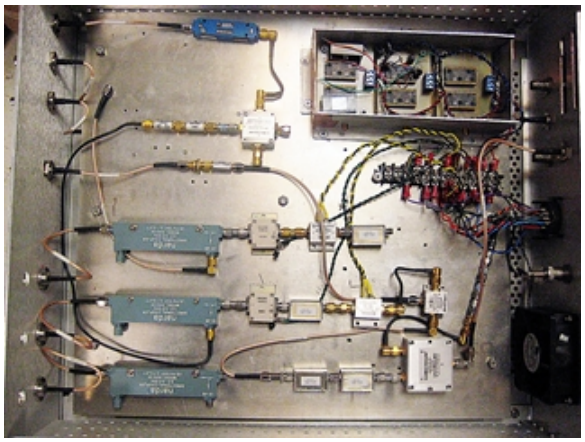
```
164     x=x(1)-1 ; find the number of elements in ls, and then we adquire the last one
165     file_name=ls(x)
166     file_name=strmid(file_name, 5, 21)
167     print, 'the name of the last .fig file is: ',file_name
168
169     lun_disp=12
170     close, lun_disp
171     spawn, "idl run_display_realtime", unit=lun_disp
172
173     end
174
175   else: return
176   endcase
177
178   return
179
180   END
```



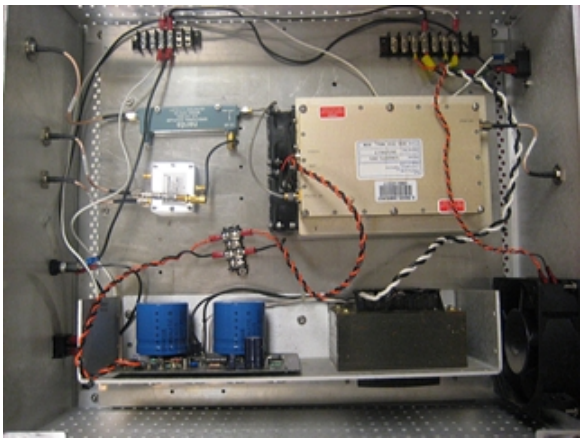
C.1 Boxes



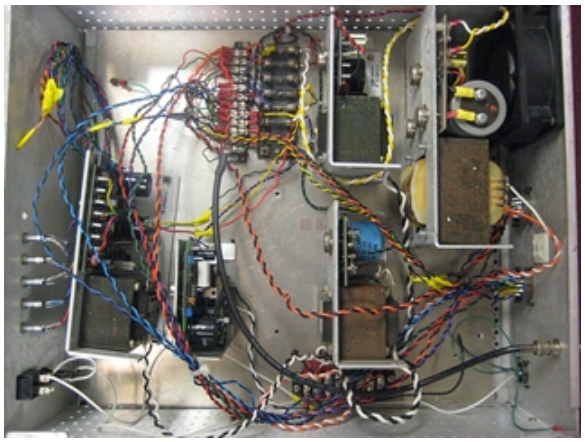
(a) FPGA Box



(b) RF Box



(c) Big Amplifier Box



(d) Power supply Box

Figure C.1: Wind Profiler Boxes

C.2 Connections

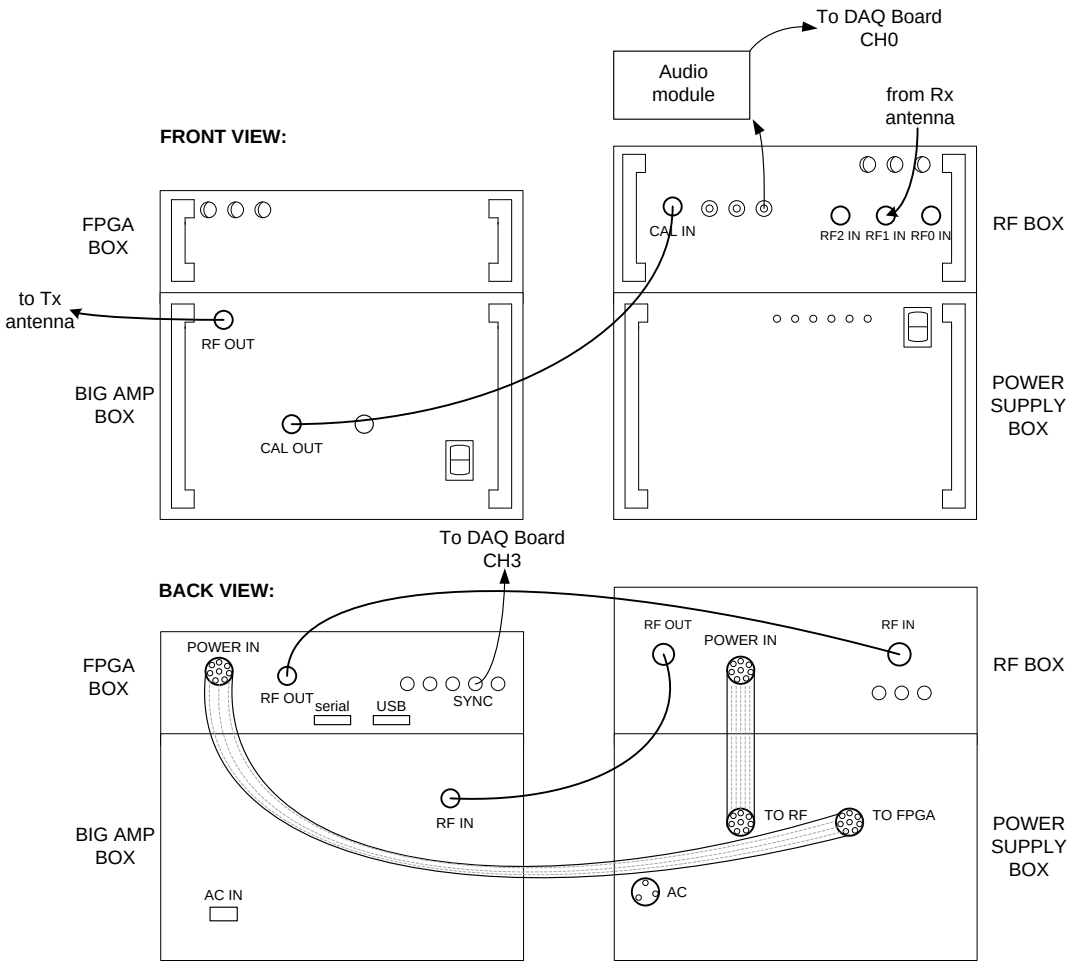


Figure C.2: Radar connections